

**RL-TR-96-9**  
**Final Technical Report**  
**February 1996**



# **ROME LABORATORY INTEGRATED DIAGNOSTIC (ID) WORKBENCH**

**IIT Research Institute**

**C. Richard Unkle**

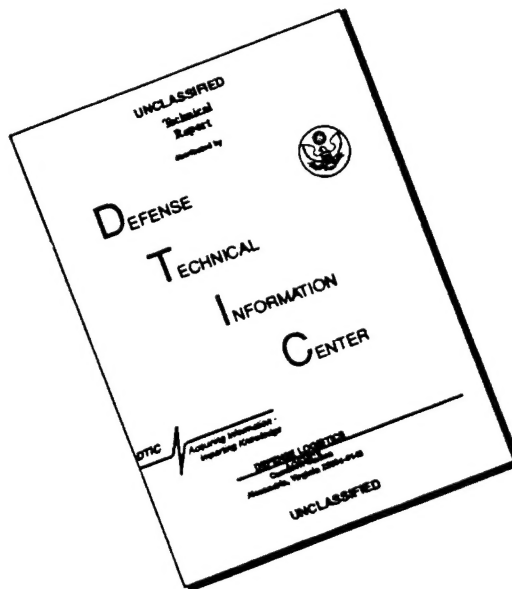
**DTIC QUALITY INSPECTED 2**

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

**19960611 064**

**Rome Laboratory  
Air Force Materiel Command  
Rome, New York**

# DISCLAIMER NOTICE

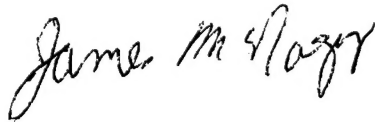


THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be releasable to the general public, including foreign nations.

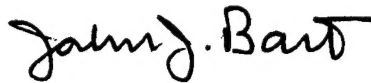
RL-TR-96-9 has been reviewed and is approved for publication.

APPROVED:



JAMES M. NAGY  
Project Engineer

FOR THE COMMANDER:



JOHN J. BART  
Chief Scientist, Reliability Sciences  
Electromagnetics & Reliability Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify Rome Laboratory/ ( ERDD ), Rome NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

# REPORT DOCUMENTATION PAGE

Form Approved  
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE February 1996		3. REPORT TYPE AND DATES COVERED Final Feb 95 - Apr 95	
4. TITLE AND SUBTITLE  ROME LABORATORY INTEGRATED DIAGNOSTIC (ID) WORKBENCH				5. FUNDING NUMBERS  C - F30602-94-C-0087 PE - 65802S PR - 6528 TA - 01 WU - 13	
6. AUTHOR(S)  C. Richard Unkle					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) IIT Research Institute Reliability Analysis Center P.O. Box 4700 Rome NY 13442-4700				8. PERFORMING ORGANIZATION REPORT NUMBER  N/A	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory/ERDD 525 Brooks Rd Rome NY 13441-4505				10. SPONSORING/MONITORING AGENCY REPORT NUMBER  RL-TR-96-9	
11. SUPPLEMENTARY NOTES  Rome Laboratory Project Engineer: James M. Nagy/ERDD/(315) 330-2241					
12a. DISTRIBUTION/AVAILABILITY STATEMENT  Approved for public release; distribution unlimited.				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words)  Integrated Diagnostics help reduce costs and improve mobility and mission effectiveness. Integrated diagnostics start when a product is first designed and developed, and conclude only when the product is no longer in use. The integrated diagnostic workbench automates how a test program set (TPS) is developed. A TPS is used on an automated test system (ATS) to verify and diagnose circuits, modules, and systems against a set of expected responses induced by a forced stimulus. This set of expected responses and a forced stimulus is referred to as a vector set. The purpose of such an integration effort is to facilitate more efficient and cost effective development and documentation of test programs for electronic systems and equipment. The goal of these efforts is to reduce both the initial and rehost generation time and cost of TPSs, and to allow greater flexibility when workload is determined. The Rome Laboratory Integrated Diagnostic (ID) Workbench is a set of integrated diagnostic tools executed from a common graphical user interface (GUI). The tools that compose the workbench are broken into two main categories, test automation and vector translation.					
14. SUBJECT TERMS  Diagnostics, Testing, Integrated diagnostics, Workbench				15. NUMBER OF PAGES 88	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT  SAR		



## TABLE OF CONTENTS

1.0	Introduction	1
2.0	Toolset Overview	5
2.1	WAVES Testbench Writer Program	5
2.2	Vector Translation Tools	6
3.0	Operating the ID Workbench	11
3.1	Tool Operation	11
3.2	Tool Descriptions	18
4.0	An Example	28
4.1	Generating a Board-Level TPS for the GenRad 2751	28
	References	39
	Appendix A Tester Independent Support Software System (TISSS)	A-1
	Appendix B The TISSS Toolset (T2)	B-1
	Appendix C Example Figures and Files	C-1

## LIST OF TABLES AND FIGURES

TABLE 1.1	DIAGNOSTIC TOOLS CURRENTLY PART OF THE RL ID WORKBENCH.....	1
TABLE 2.1	WAVES TESTBENCH WRITER INPUT FILES.....	5
TABLE 3.1-1	ID WORKBENCH FUNCTIONS.....	18
TABLE B1	.POW FILE FIELD DESCRIPTIONS AND RULES.....	B-5
TABLE B2	DEVICE PIN CROSS REFERENCE LIST .....	B-8
FIGURE 1.1	CURRENT TOOLSET OPERATION FLOW .....	2
FIGURE 2.1	WAVES TESTBENCH WRITER INPUT AND OUTPUT .....	5
FIGURE 2.2	VECTOR TRANSLATION TOOL INPUT AND OUTPUT .....	7
FIGURE 2.3	SAMPLE TRF FOR A D-FLIP FLOP DEVICE .....	8
FIGURE 2.4	TRF-TO-TDL & IN-STEP INPUT AND OUTPUT .....	10
FIGURE 3.1	ID WORKBENCH MAIN SCREEN .....	12
FIGURE 3.2	EXAMPLE TOOL INPUT SCREEN .....	13
FIGURE 3.3	SELECTING A FILE .....	14
FIGURE 3.4	EXAMPLE OUTPUT MESSAGES FOR THE GENERATE WAVES TESTBENCH TOOL .....	15
FIGURE 3.5	OUTPUT FILE VIEWING WINDOW .....	16
FIGURE 3.6	EXPORT FILE SELECTION SCREEN .....	17
FIGURE 3.8	GENERATE WAVES TESTBENCH INPUT SCREEN .....	20
FIGURE 3.9	GENERATE WAVES FLATTENER INPUT SCREEN .....	21
FIGURE 3.10	GENERATE PARTIAL LOOKUP TABLE INPUT SCREEN .....	22
FIGURE 3.11	GENERATE VECTOR DATABASE INPUT SCREEN .....	24
FIGURE 3.12	SELECTING THE OUTPUT OF THE TESTBENCH SIMULATION FILENAME .....	25
FIGURE 3.13	INITIAL INPUT SCREEN FOR GR2751 TRANSLATOR (TGO) TOOL.....	26
FIGURE 3.14	INITIAL INPUT SCREEN FOR GENERATE TDL TOOL .....	27
FIGURE 4.1	PARTIAL LOOKUP TABLE SCREEN .....	30
FIGURE 4.2	OUTPUT FILE SELECTION SCREEN .....	31
FIGURE 4.3	TEXT EDITOR SCREEN .....	32
FIGURE 4.4	WAVES TESTBENCH GENERATOR SCREEN .....	33
FIGURE 4.5	WAVES TESTBENCH OUTPUT FILES .....	34
FIGURE 4.6	GENERATE VECTOR DATABASE SCREEN .....	35
FIGURE 4.7	GR2751 TRANSLATOR SCREEN .....	36
FIGURE 4.8	VECTOR TRANSLATION OUTPUT FILES .....	37
FIGURE 4.9	GENERATE TDL SCREEN .....	38
FIGURE A1	EXAMPLE OF A TISSS ENVIRONMENT .....	A-2
FIGURE A2	EXAMPLE IN-STEP EXECUTION .....	A-4
FIGURE A3	EXAMPLE TEST AUTOMATION PROCESS USING TISSS .....	A-5
FIGURE B1	T <sup>2</sup> I/O REQUIREMENTS FOR THE GENRAD 2751 .....	B-2
FIGURE B2	EXECUTION OF VECTOR TRANSLATION TOOL "Generate Vector Database" .....	B-11
FIGURE C1	EXAMPLE CIRCUIT .....	C-2
FIGURE C1.1	VHDL STRUCTURAL DESCRIPTION FOR EXAMPLE CIRCUIT .....	C-3

## 1.0 INTRODUCTION

The Rome Laboratory (RL) Integrated Diagnostic (ID) Workbench is part of an on-going effort at RL to develop and integrate diagnostic tools and techniques. The purpose of such integration efforts is to facilitate more efficient and cost effective development and documentation of test programs for electronic systems and equipment. The RL ID Workbench is a prototype package that allows multiple tools to be executed from a common graphical user interface (GUI). The tools that are currently part of the workbench are listed in Table 1.1.

**TABLE 1.1: DIAGNOSTIC TOOLS CURRENTLY PART OF THE RL ID WORKBENCH**

TOOL NAME	ID WORKBENCH OPTION	DESCRIPTION
WAVES TESTBENCH WRITER	<u>Test Automation Tool</u> - GENERATE WAVES TESTBENCH	Currently takes a VHDL structural only file of a printed circuit board and develops a VHDL simulation testbench, with nodal analysis capabilities, compatible with the Waveform and Vector Exchange Specification (WAVES) (IEEE Std. 1029.1991).
TRF WRITER	<u>Vector Translation Tool</u> - GENERATE VECTOR DATABASE	Develops a test requirements file(TRF) from information contained in the output of the VHDL/WAVES testbench simulation and a file containing the VHDL signal names cross-referenced to actual device pin names
TRF-TO-TDL TRANSLATOR	<u>Vector Translation Tool</u> - GENERATE TDL	Translates a TRF into a Test Description Language (TDL) file compatible with the IN-STEP program
VECTOR TRANSLATOR	<u>Vector Translation Tool</u> - GENERATE VECTOR DATABASE - GR2751 TRANSLATOR	Translates digital test vectors captured as output of the testbench referred to above, into vector files compatible with the <u>Genisys</u> <sup>TM</sup> TPS development toolset from GenRad Inc.

In addition to the above listed tools, the ID Workbench contains other utilities that are explained in Section 3.0 and in Appendixes A-C.

Figure 1.1 provides an overview of how the tools are currently designed to function. The toolset depicted in Figure 1.1 was designed to aid in the development of a test program set (TPS) for digital printed circuit boards whose design information is captured in the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL). The tools are also designed to work with Unit Under Test (UUT) digital vector information captured in the Waveform and Vector Exchange Specification (WAVES), a sister standard to IEEE Standard 1076, VHDL.

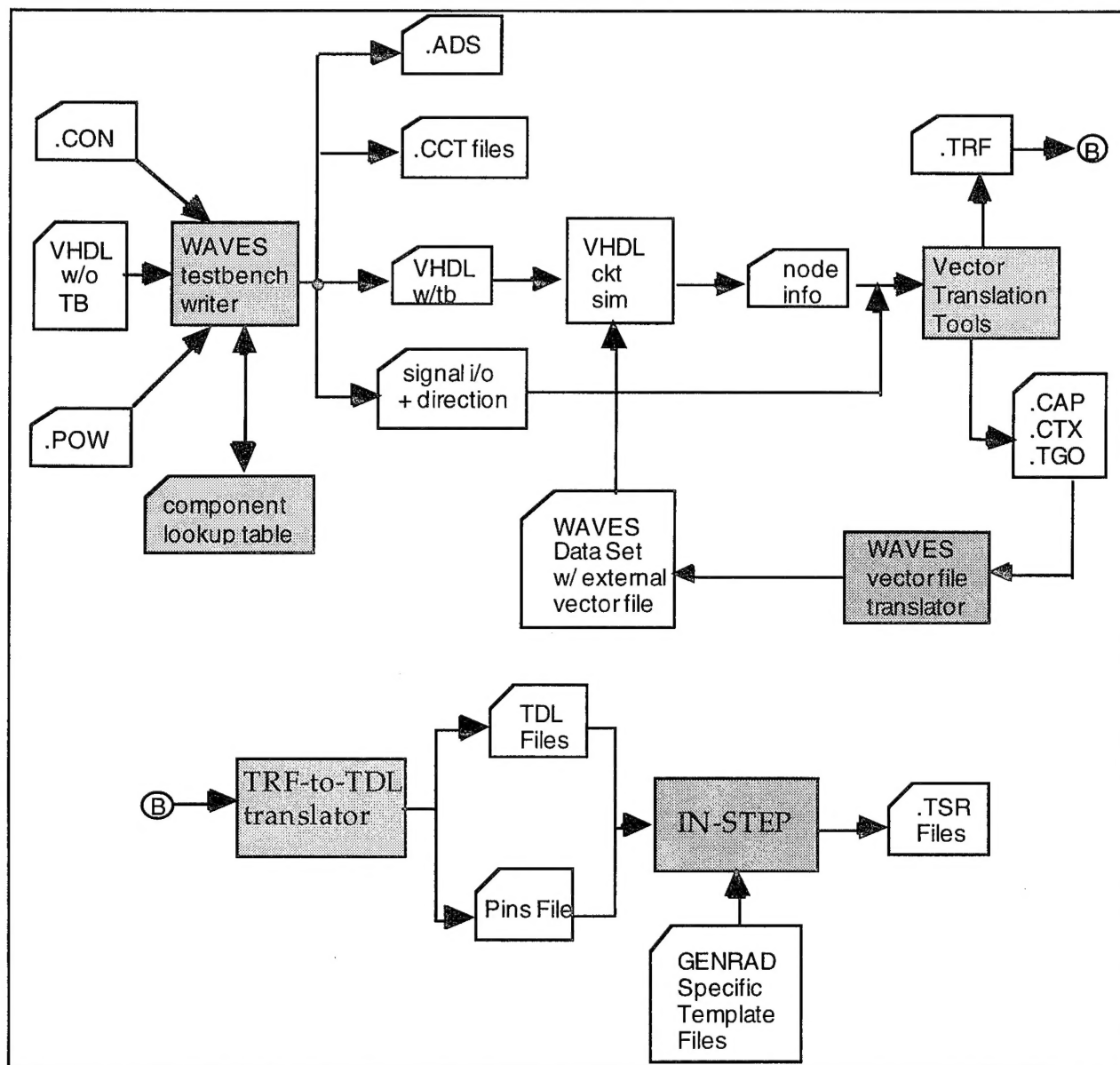


FIGURE 1.1: CURRENT TOOLSET OPERATION FLOW

Much of the process is tester independent (i.e., the tools are not designed with any particular tester file formats in mind). However, at some point in time, the tester being used to host the TPS must be considered. In this case, the current set of tools are compatible with a GenRad 275X series board tester. Figure 1.1 reflects the file formats associated with the GenRad tester. These formats are explained in more detail in Section 2.0, Toolset Overview.

The current set of tools integrated within the ID Workbench are designed to operate as depicted in the flow diagram of Figure 1.1. The initial step in the process is to create a VHDL testbench using a VHDL structural model as input. Other files shown (i.e., .CON, .POW, etc.) as inputs to the WAVES testbench writer tool are needed for development of test vector information and files required for test by most ATE. The .CON file is an ASCII file that identifies the size of the board's physical connector, and maps the connector pin numbers to the pin names. The pin names are external signal names contained in the VHDL file (s). The .POW file is another ASCII file that contains a list of all voltage supply signals used on the board. In addition, the nominal voltage values, maximum current values, and ground signal name that corresponds to each voltage signal are also listed in this file.

The final input, "component lookup table" is a lookup table for each of the individual components contained in the VHDL structural file. This file provides a list of each component's pin name, pin number, and pin direction. The pin name is the generic component pin name, and the pin number is the terminal number. Each of the three input files described, .CON, .POW and component lookup, are necessary to create the GenRad specific files shown in Figure 1.1 (.ADS, .CCT). These files are required by the tester, and similar files are typically needed by all testers. Further details on the input files described can be found in Appendix B.

The primary output of the WAVES testbench writer is a simulatable testbench that uses the stimulus information represented in the WAVES format. Additional outputs of the tool, as shown in the figure are required for other down stream processes or by the target ATE. The ID workbench does contain a utility that will develop a partial lookup table from the VHDL input file. The partial table is then completed manually using a text editor.

The testbench is designed to capture test vector information in a generic format. Output information, as well as internal nodal data (used for diagnostics) are captured from the testbench in an RL develop standard format, including any required timing information. This data is then used by the Vector Translation tools to develop tester specific test vector files. An additional output of these tools is the test requirement file (TRF) that is used to create a complete set of TDL files for the UUT using the TRF-to-TDL translator. The final step in the process is to use the TDL information to create the remaining ATE test files using ATE specific template files and the IN-STEP program. The IN-STEP is not currently part of the ID Workbench, but can be executed separately .

Another tool shown in Figure 1.1, but also not yet part of the ID workbench, is the WAVES Vector File Translator. This tool currently takes a vector file in the GenRad 275X format, and creates a WAVES formatted external vector file that can be used to drive the VHDL/WAVES testbench. This tool currently runs on a PC, the primary reason that it is not part of the current version of the ID Workbench. It is not known at this time if there are plans to port this tool to the SUN environment.

## 2.0 TOOLSET OVERVIEW

### 2.1 WAVES Testbench Writer Program

This program is currently designed to generate several outputs, given the specific input files previously described and repeated in Figure 2.1. A list of the required input files for this program, and their description is given in Table 2.1

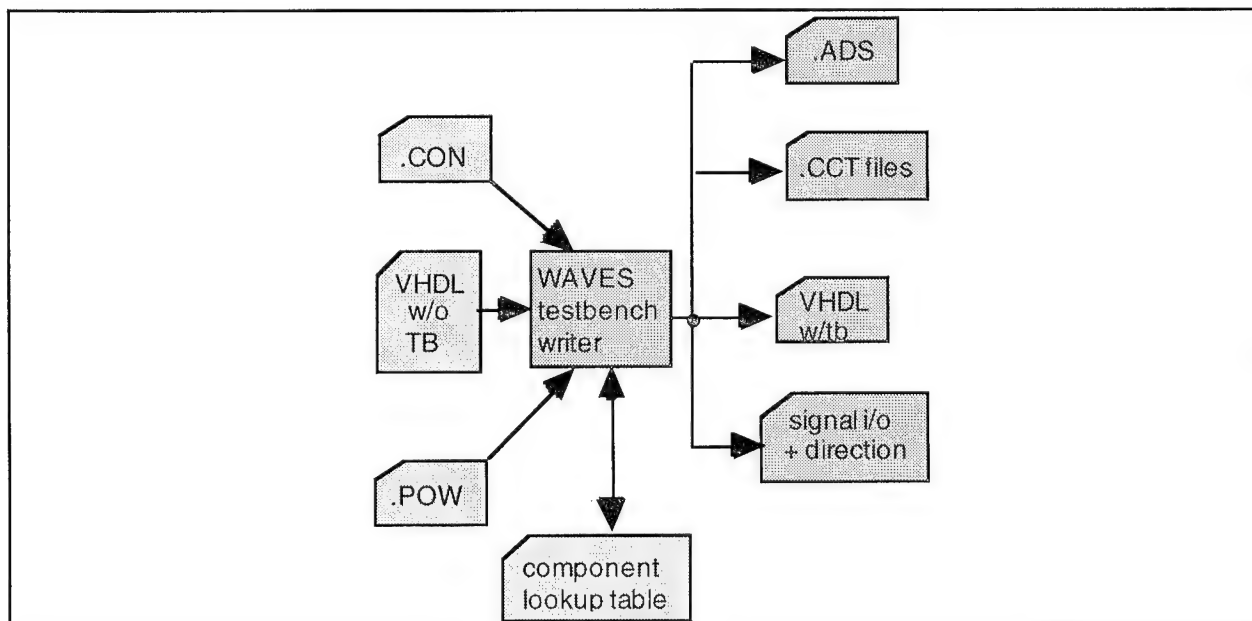


FIGURE 2.1: WAVES TESTBENCH WRITER INPUT AND OUTPUT

TABLE 2.1: WAVES TESTBENCH WRITER INPUT FILES

INPUT FILE	DESCRIPTION
VHDL W/O TB	A structural only VHDL description file of a UUT
.CON	A file listing the UUT connector type, number of pins, and pin names
.POW	A file that lists the UUT voltage supply values, including corresponding current values and ground signal names
.TBL	A file listing each UUT component and all generic pin names, pin numbers, and pin I/O direction

The connector, power, and component files are currently required to create GenRad 2751 specific files listed as '.ADS' and '.CCT' in Figure 2.1. These files are used by the GenRad Genisys™ tools in creating a TPS for the GenRad tester. It is assumed that

similar files will be needed by other testers and, therefore, the .CON, .POW, and .TBL input files will be relevant to several testers, although slight modifications will be needed from tester to tester.

The primary output of the WAVES Testbench Writer tool is a WAVES compatible VHDL testbench that can be used to develop test vectors for the UUT. The testbench created is compatible with vector information contained in the WAVES formats, using an external file for the raw vector data. The testbench, noted as 'VHDL w/tb' in Figure 2.1 needs to be simulated by a VHDL simulator to produce the necessary vector information.

One additional file created by the WAVES TESTBENCH WRITER is the 'signal i/o + direction' file that is used in a down stream process to help translate vector information produced during simulation to a tester specific format (GenRad 275X in this case). This file is also used to aid in the creation of the TRF for the UUT. The name of the 'signal i/o + direction' file is <filename>.sig, where <filename> is the same as the VHDL input file.

Appendix A-C provides more detailed information on both input and output file formats.

## 2.2 Vector Translation Tools

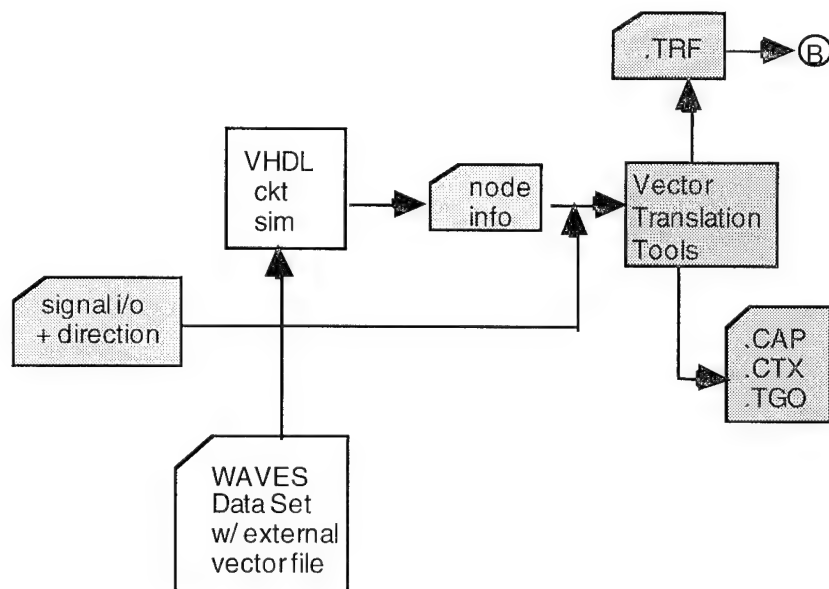
The Vector Translation Tools are listed below. Each of these tools are described within the following subsections of the report.

- Vector Translator
- TRF Writer
- TRF-to-TDL Translator

### 2.2.1 Vector Translator

The tool known as the Vector Translator tool is really two processes combined into a single tool. One process is called "tb\_to\_nodes", and the other process is called "nodes\_to\_tgo". The flow diagram for this tool is shown in Figure 2.2.





**FIGURE 2.2: VECTOR TRANSLATION TOOL INPUT AND OUTPUT**

#### 2.2.1.1 tb\_to\_nodes and nodes\_to\_tgo

The `tb_to_nodes` process creates two different outputs from UUT nodal information captured via simulation of the testbench described previously. The other input is the signal i/o + direction file, also described previously. One output is the input file required by the `nodes_to_tgo` process, and the other is a probing or diagnostics database to be used for troubleshooting. The probing data is written in the GenRad 275X tester format. This file is called the `<filename>.cap` file, where `<filename>` is the name of the input file (nodal information file) to the Vector Translator tool. The second output, which is the input to the `nodes_to_tgo` process, is called `<filename>.nodes`. The "nodes\_to\_tgo" process translates primary i/o information captured via simulation of the VHDL/WAVES testbench, into a format called "tgo" which is the test vector format required by the GenRad 275X board tester.

#### 2.2.2 TRF Writer

The TRF Writer utility creates a test requirements file (TRF) based on information contained in the connector and power input files described above, as well as the vector file generated during simulation of the VHDL/WAVES testbench. The TRF is an intermediate format file that contains information relevant to testing electrical parameters, such as current and voltage, of a UUT. The TRF is used to generate a Test Description Language (TDL) file and 'PINS' file that are required as input to the IN-STEP tool. (Note that the IN-STEP tool is not currently part of the ID WORKBENCH interface program.) An example TRF is provided below as Figure 2.3. See reference [1] for more information on the TRF.

#### FILES:

```
-- The .tdl and .pins files will have the same base name as this .trf file
trf = master; -- (in) master.trf.xxx are the TDL templates
waves = sample_waves; -- WAVES data set referenced in TDL
```

#### VALUES: -- symbolic values for TDL

```
NUM_VEC = 100; -- needed for TDL templates

VIL1 = 0.7 v; -- TDL does not permit
VIH1 = 2.0 v; -- more than one value for VIL, VIH
IIL1 = 40 ua;
IIL1 = -0.8 ma;

VOH1 = 2.5 v;
VOL1 = 0.4 v;
IOH1 = -400 ua;
IOL1 = 4.0 ma;

-- Positive supply:
VCC1 = 5 v;
MAX_VCC1 = 5.5 v;
MIN_VCC1 = 4.5 v;

ICC1 = 8 ma;
MAX_ICC1 = 40 ma; -- about five times the spec value

VGG = 0 v;
MAX_VGG = 0 v;
MIN_VGG = 0 v;

IGG = -10 a;
MAX_IGG = -10 a;
```

#### DEFAULTS:

```
-- The defaults *must* use the symbolic values
-- defined above in the "VALUES:" section.
-- That is, in this section use statements such
-- as "VOH = VOH1" instead of "VOH = 3.0 V".

-- Values for VOH, VOL, IOH, IOL, VIH, VIL
-- *must* be provided as defaults because they are used
-- in the xxx.TRF.MAIN file.

VOH = VOH1; -- every pin gets this, if specified,
VOL = VOL1; -- but the defaults can be overridden
VIL = VIL1;
VIH = VIH1;
```

FIGURE 2.3. SAMPLE TRF FOR A D-FLIP FLOP DEVICE

```
IIL = IIL1;  
IIH = IIH1;  
IOH = IOH1;  
IOL = IOL1;
```

PINS:

```
name = CLR1, type = INPUT, pin = 1, term = 1;  
name = PRE1, type = INPUT, pin = 2, term = 4;  
name = D1, type = INPUT, pin = 3, term = 2;  
name = CLK1, type = INPUT, pin = 4, term = 3;  
name = CLR2, type = INPUT, pin = 5, term = 13;  
name = PRE2, type = INPUT, pin = 6, term = 10;  
name = D2, type = INPUT, pin = 7, term = 12;  
name = CLK2, type = INPUT, pin = 8, term = 11;  
name = Q1, type = OUTPUT, pin = 9, term = 5;  
name = QB1, type = OUTPUT, pin = 10, term = 6;  
name = Q2, type = OUTPUT, pin = 11, term = 9;  
name = QB2, type = OUTPUT, pin = 12, term = 8;  
name = GND, type = POWER, pin = 0, term = 7,
```

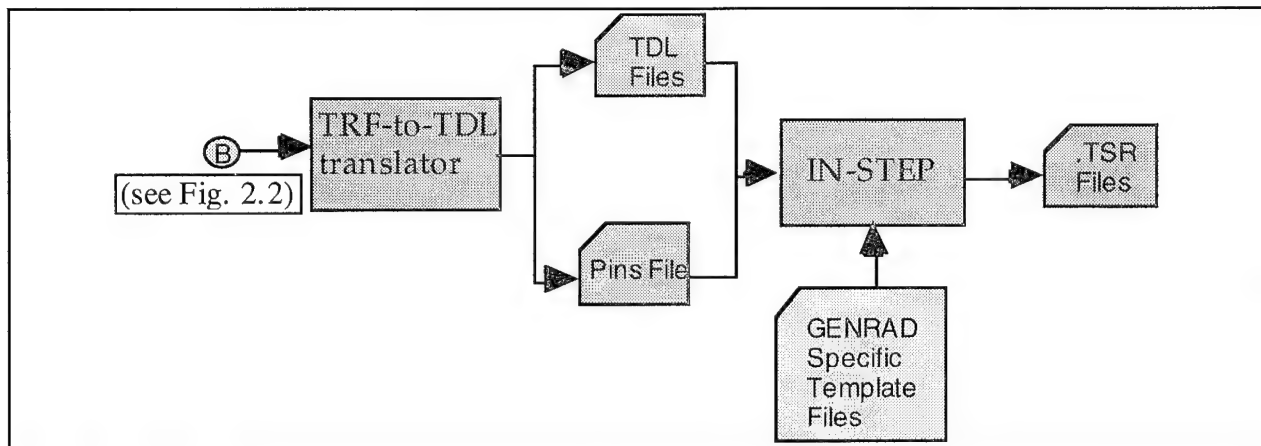
```
supply_voltage = VGG,  
  max_supply_voltage = MAX_VGG, min_supply_voltage = MIN_VGG,  
supply_current = IGG,  
  max_supply_current = MAX_IGG;
```

```
name = VCC, type = POWER, pin = 0, term = 14,  
  supply_voltage = VCC1,  
    max_supply_voltage = MAX_VCC1, min_supply_voltage = MIN_VCC1,  
  supply_current = ICC1,  
    max_supply_current = MAX_ICC1;
```

**FIGURE 2.3. SAMPLE TRF FOR A D-FLIP FLOP DEVICE - CONTINUED**

### 2.2.3 TRF-TO-TDL Translator

The TRF-to-TDL Translator tool takes the information captured in the TRF, and produces several files, as shown in Figure 2.4 below. These files, which collectively form a TDL description file, when combined with specific ATE template files, are used by the IN-STEP to produce tester specific test program files. A brief explanation of TDL is provided here. A more complete description can be found in reference [2].



**FIGURE 2.4. TRF-TO-TDL & IN-STEP INPUT AND OUTPUT**

### 2.2.3.1 TDL

The Test Description Language (TDL) is a non-proprietary data format that was developed during early stages of RL's Tester Independent Support Software System (TISSS) program as a means for capturing digital component test requirements information, such as absolute maximum ratings for current, voltage, etc., power conditions, drive/load conditions, and recommended operating conditions. While the TDL describes test requirements and test specifications, it is not intended to provide descriptions of test methodologies. Capture of test requirements, specifications, sequencing, and intent is supported via two high level structures: the test philosophy specification and the test instantiation specification. The test philosophy specification is used to describe sets of tests used for the validation units and unit classes within a particular technology type (e.g., TTL, CMOS, etc.). The test instantiation specification is a hierarchical structure that contains two main parts, the pin definition specification and the test plan build specification. The pin definition specification describes the pin characteristics of all pins on the UUT and groups UUT pins into pin sets. The test plan build specification specifies the tests and sets of tests, their sequencing and all parametric data that are required to validate the UUT according to one of the test qualifications in what is known as the Electrical Test Requirements table, found in the test philosophy file described previously.

### 3.0 OPERATING THE ID WORKBENCH

The ID Workbench software tool currently runs on a SUN Sparc workstation, and requires X-WINDOWS to operate. The executable file for the ID Workbench is called 'wkbench.x', and should be installed in an appropriate directory on the users workstation. Once the executable is installed, the following set environment statements must be added to the user's '.cshrc' file:

```
WORKBENCH_EXES_LOCATION=/<directory path>  
WORKBENCH_TEXT_EDITOR=textedit
```

In the first command above, <directory path> is the complete directory path to the location of the wkbench.x executable file. The following is provided as an example:

```
WORKBENCH_EXES_LOCATION=/home/vortex/elefante/WORKBENCH/  
RUNTIME
```

The second command above is needed to define the text editor that the ID Workbench program will use to display and modify files created by the various tools described herein.

Given that the user is logged onto the SUN system and is in the directory containing the ID Workbench executable program, simply type 'wkbench.x' and press enter. The main screen shown in Figure 3.1 should appear as a result. From the main screen, three options are available: Test Automation Tools, Vector Translation Tools, and Exit.

#### 3.1 TOOL OPERATION

Clicking on either the Test Automation Tools option or the Vector Translation Tools option will produce a list of tools that can be executed from the ID Workbench. For instance, clicking on Test Automation Tools presents the following list of tools:

- Generate WAVES Testbench
- Generate WAVES Flattener
- Generate Probe Database
- Generate Partial Lookup Table
- Generate GENRAD CKT file

Clicking on any listed tool produces the screen shown in Figure 3.2, which shows the "Generate WAVES Testbench" as an example. The screen will show the input files that are required to execute the tool.

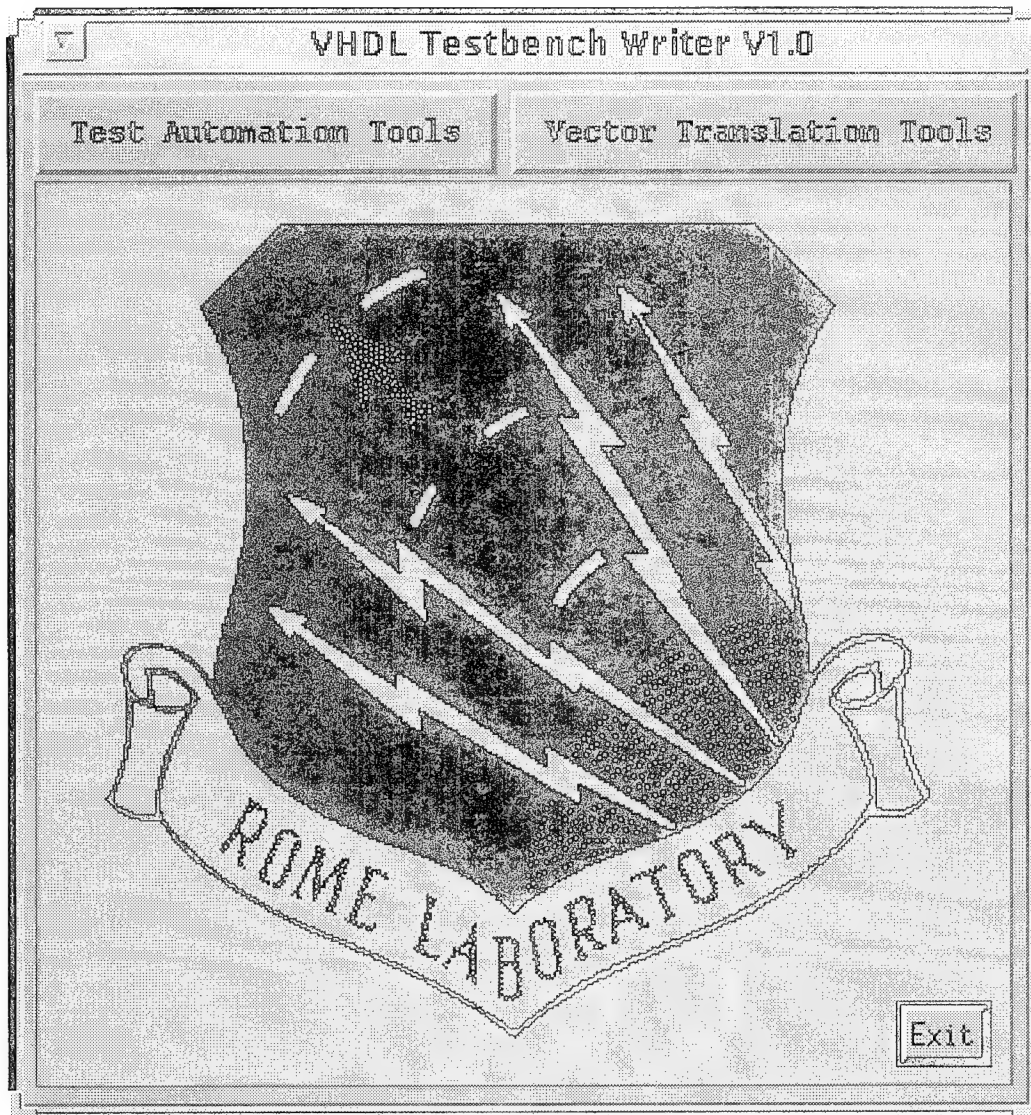


FIGURE 3.1: ID WORKBENCH MAIN SCREEN

Clicking on any of the "Browse" buttons will bring up a window showing a list of files in the current directory from which to choose. The user may also type in the directory path and filename. The full directory path is required as the ID Workbench makes no assumptions as to where files exist, while allowing them to exist in any user directory and/or sub directory. For each of the file selections (e.g., connector, lookup table, power and VHDL), a filter has been set to show only those files with the expected extension. For the connector filename, for example, all files with a '.con' extension are displayed. Likewise for the lookup table filename, files with a '.tbl' extension are displayed, '.pow' for the power filename, and '.vhd' for the VHDL filename. All extensions are in lower case. An example of what is displayed when the Browse button for the power filename is selected is shown in Figure 3.3.



Generate WAVES Testbench

The connector filename

The lookup table filename

The power filename

The VHDL filename

Output Messages Generated

Error Messages Generated

FIGURE 3.2: EXAMPLE TOOL INPUT SCREEN

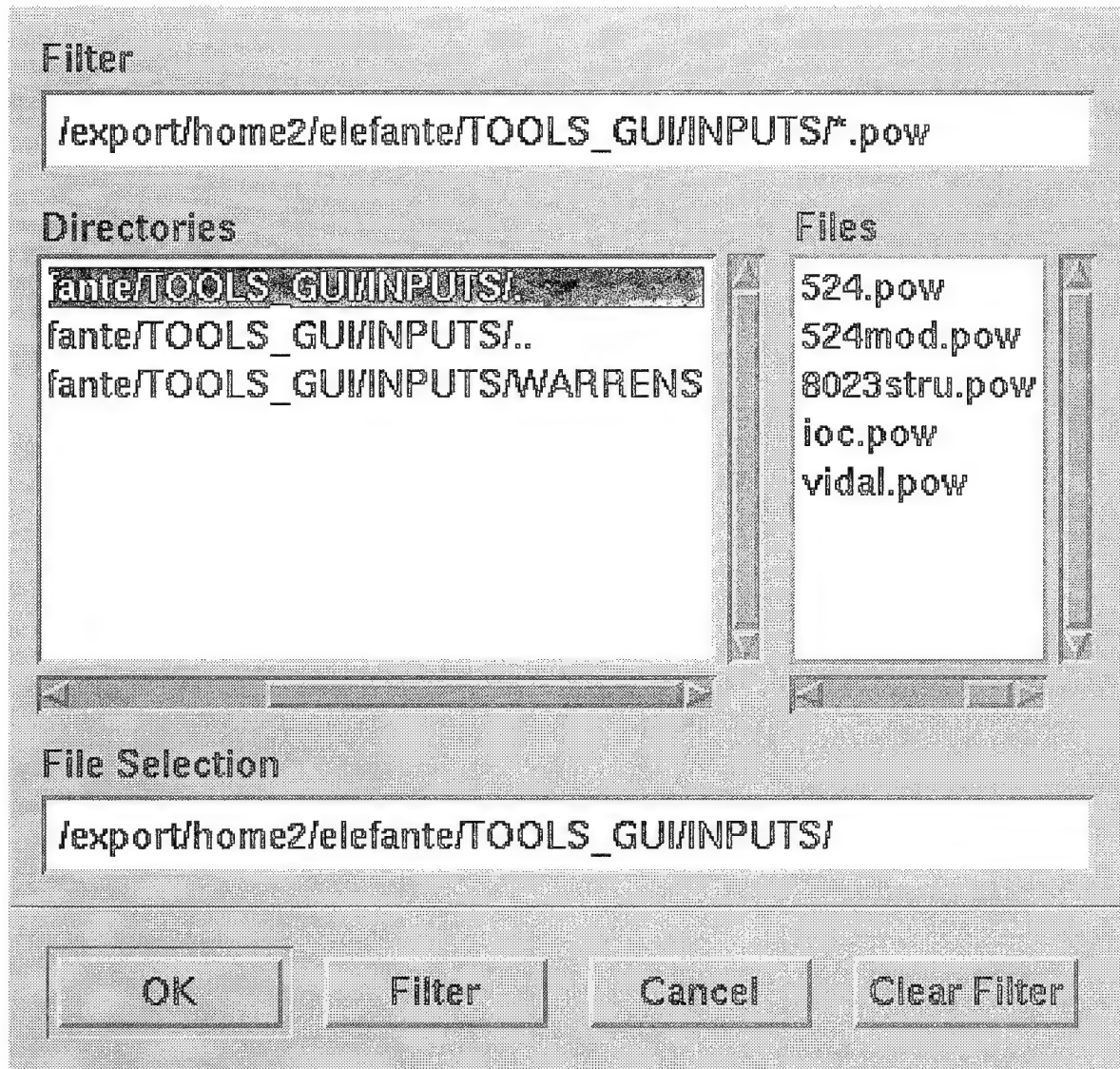


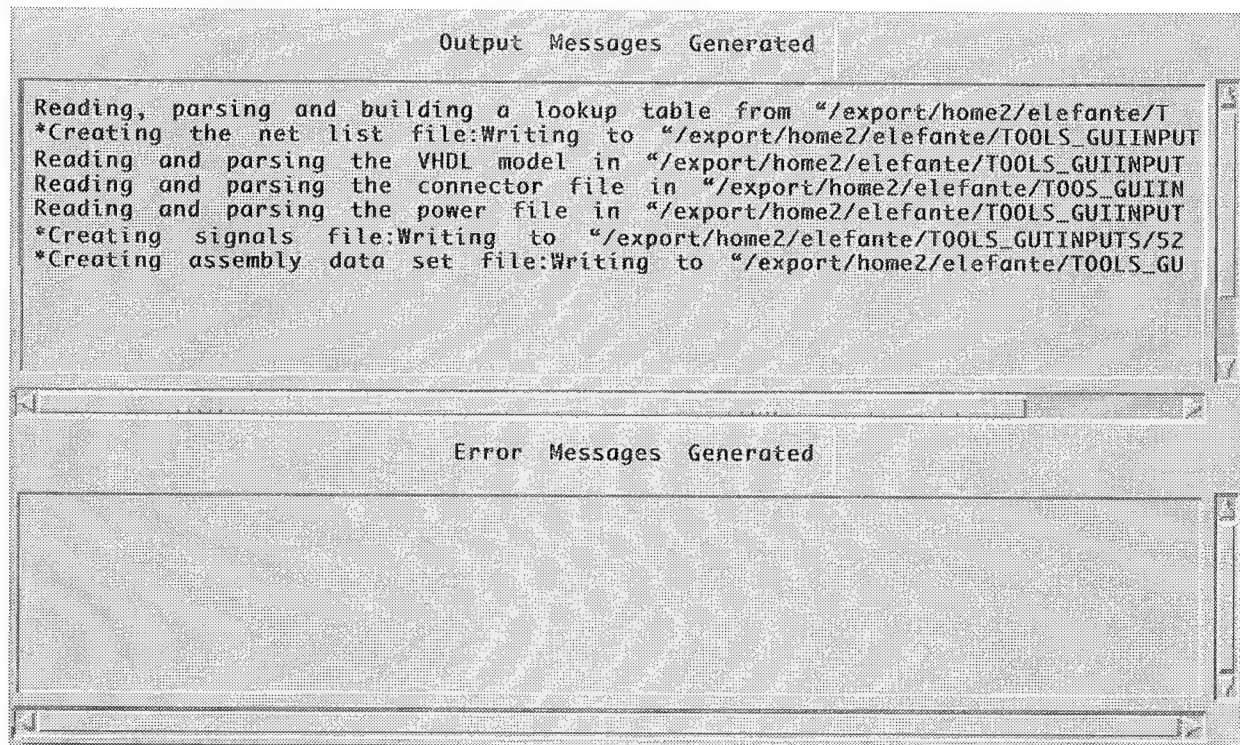
FIGURE 3.3: SELECTING A FILE

The filter of “\*.pow” is being applied to the current working directory which is /export/home2/elefante/TOOLS\_GUI/INPUTS for this example. If there are no files with the .pow extension, then the user must either change directories until the correct file is found, or the filename may not have the .pow extension. In this latter case, clicking on the “Clear Filter” button will enable the user to see all files contained within this directory. Note in the Figure that the current directory being viewed is indicated by the single period ending. To move one level above this, double click on the directory name having two periods at the end (i.e., /export/home2/elefante/TOOLS\_GUI/INPUTS/..). Changing to any of the other directories listed is also possible by double clicking the mouse while the mouse pointer is on the directory name, or by highlighting the directory name and clicking the OK button. Once the file desired is listed in the files window, double click on the



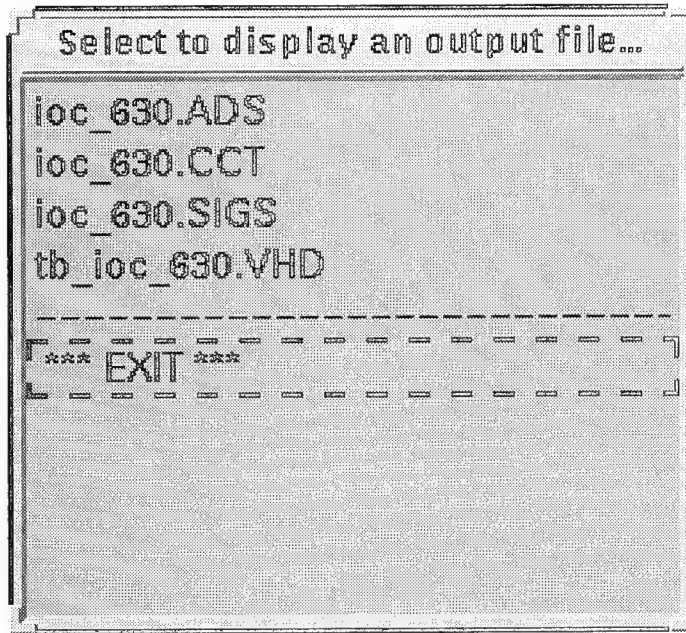
file name, or highlight it with a single click and click on the OK button to select it. To reset the Filter, simply click on the Filter button.

Once all of the appropriate files have been selected, click on the "Run the translation" button to execute the selected tool. If there are no errors, information will appear in the "Output Messages Generated" window shown in Figure 3.1. An example of this for the Generate WAVES Testbench tool is provided in Figure 3.4. If errors do occur, any error messages generated are captured and presented in the Error Messages Generated window. Upon completion of any one of the tools, the window shown in Figure 3.5 will appear that enables viewing of all output files generated by the tool being used.



**FIGURE 3.4. EXAMPLE OUTPUT MESSAGES FOR THE GENERATE WAVES TESTBENCH TOOL**

As an example, Figure 3.5 shows the files that were generated by the Generate WAVES Testbench tool for VHDL input file "ioc\_630.vhd". Clicking on any of the files listed will automatically put the file in a text editor where it can be viewed and/or modified as necessary. Once exiting out of the editor, click on EXIT, where shown in the above figure, to return to the main program screen.



**FIGURE 3.5: OUTPUT FILE VIEWING WINDOW**

### 3.1.1. The Import and Export Features

Each of the main tool screens, ( Figure 3.1 for example), contains two selected items named Import and Export. The Export function allows the user to save a set of input files such that these same files can be "imported" at the filename screen from a single file, without having to re-select each file individually. The feature enables multiple runs for the same design to be executed quickly, when the same input filenames are used. To exercise the export option, simply click on the export button, after all input files have been selected. This action results in the screen shown in Figure 3.6. The user then enters a filename, with the extension '.export', on the filename line to save the input files that have been selected for a specific tool. The next time the tool is exercised, simply click on the Import button, and all files with the ".export" extension are listed in the files window as shown in the Figure 3.7. Select the filename with the desired files in it by double clicking on the filename, or by highlighting the filename and clicking the OK button. All tool input filenames are then automatically entered into the appropriate place (See Figure 3.1).

### 3.1.2 Screen Selection Summary

Table 3.1 presents a summary of each of the functions that are available when running each of the workbench tools. The table shows the name of the function, the screen where it appears, and the action taken when chosen.

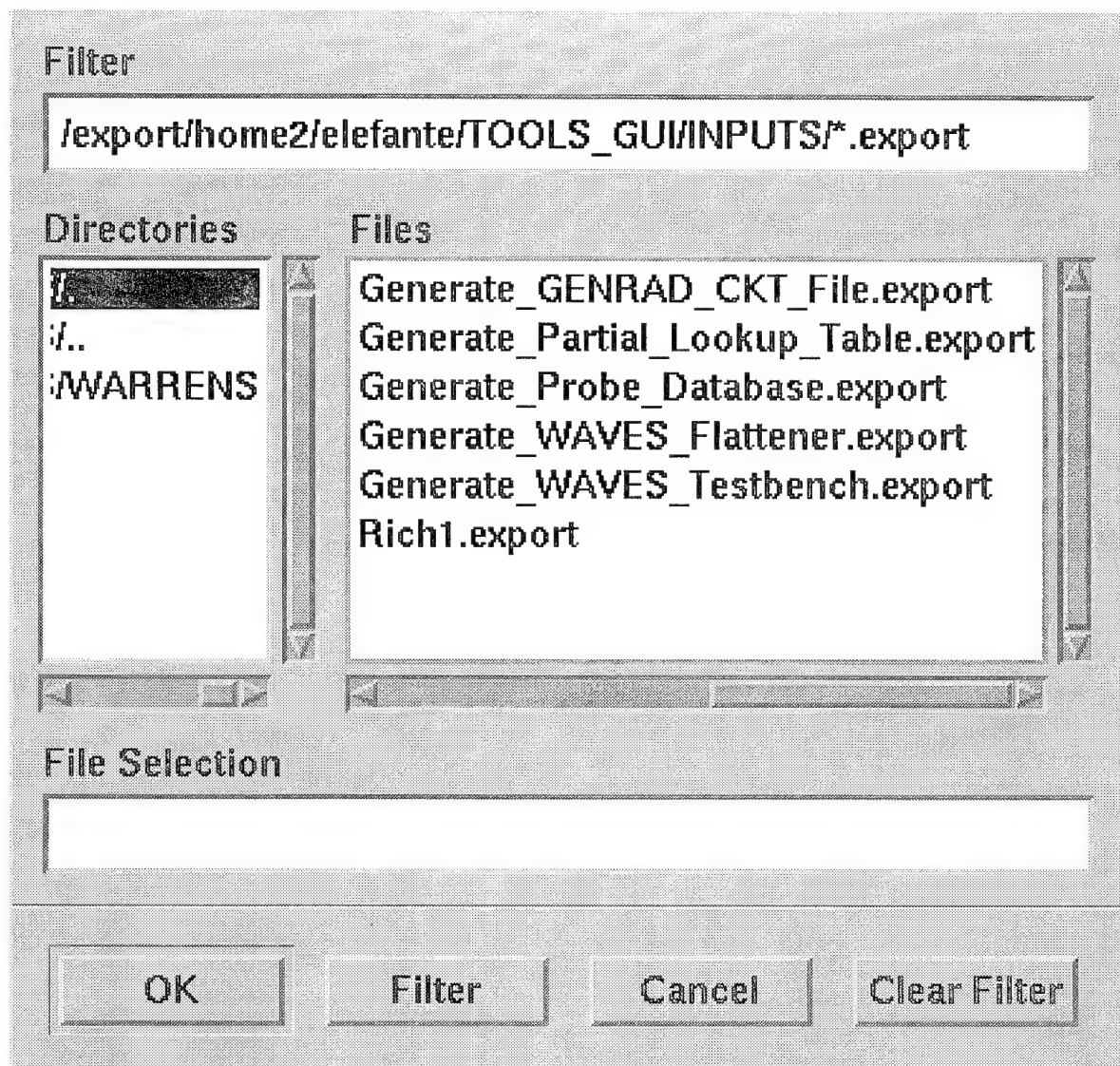


FIGURE 3.6 & 3.7: EXPORT FILE SELECTION SCREEN

**TABLE 3.1-1: ID WORKBENCH FUNCTIONS**

Function	Screen/Window	Action	Example Figure
Browse	Main Tool Screen. Used to quickly select an input filename needed to execute the tool	Presents a window showing the current working directory and the files in that directory based on the current filename extension <b>filter</b>	See Figure 3.2
Filter	Filename Filter Window	Turns the current filter back on. When the filter is on, only those filenames with the filtered extension will be presented in the files window	See Figures 3.3, 3.6 and 3.7
Clear Filter	Filename Filter Window	Turns the current browse filter off, allowing the user to view and select any/all filenames	None
Cancel	Filename Filter Window	Cancels the current operation and return to the main tool screen	See Figures 3.3, 3.6 and 3.7
OK	Filename Filter Window	Enters the highlighted filename from the files window into the main tool screen (not required when filename is chosen by double clicking )	See Figures 3.3, 3.6 and 3.7
Run Translation	All Main Tool Screens	Executes the current tool for the selected input files	See Figure 3.2
Exit	All Main Tool Screens and Main Menu Screen	Exit to Main Menu or Exit the ID Workbench	See Figure 3.1, 3.2

## 3.2 TOOL DESCRIPTIONS

Each of the tools that are executable from within the ID Workbench will be described briefly in this section. Where applicable, references to the general selections described in Section 3.1 above will be made.

### 3.2.1 Test Automation Tools

By clicking the Test Automation Tools, the following options are presented:

- Generate WAVES Testbench
- Generate WAVES Flattener
- Generate Probe Database
- Generate Partial Lookup Table
- Generate GENRAD CKT file

Generate Probe Database and Generate GENRAD CKT file are currently Alpha version tools at best. The Vector Translation Tools contain software that will produce the same results and have been more fully tested. Therefore, while these options are available, they will not be described in this report. Future updates to the ID Workbench will not include these options as part of the Test Automation Tools menu.

#### 3.2.1.1 Generate WAVES Testbench Tool

When this tool is selected, the screen in Figure 3.8 will appear. Note that four input files are required to execute this tool as described in Section 2.0 and in the flow diagram of Figure 2.2. Once each of the correct files has been entered using either the Browse button or manually by the user, then click on Run the translation, and the WAVES testbench file will be created. This file, along with the appropriate WAVES dataset files can be simulated to produce vector information needed for test. Execution of this tool and subsequent simulation of the output must be performed prior to execution of the Vector Translation Tools that are described in section 3.2.2.

#### 3.2.1.2 Generate WAVES Flattener

Some older model Automatic Test Equipment (ATE) that are still being used are less dynamic than more state-of-the-art ATE in that multiple cycle times are not possible. While all ATE require test vectors to be cyclized, many older ATE can only handle a single cycle time. In other words, every cycle would be of the same duration (e.g., 1000 nanoseconds (ns)). This tool was created, therefore, to develop a testbench that would produce test vector information in equal cycles. Currently, the testbench is written to create test cycles of 1000 ns in length. This is accomplished by capturing test vectors in the WAVES format, and then re-capturing the information using a VHDL simulator. This requires the development of a testbench. Note that the WAVES Flattener testbench would be executed to produce the test vector file used as input to the simulation of the testbench produced from the Generate WAVES Testbench tool. By doing this, the output of the WAVES testbench will maintain the integrity of the probe database.

The set up and use of this tool within the ID Workbench is exactly the same as described in Section 3.1 and 3.1.1. Figure 3.9 shows the tool input screen. Note that the only difference between this tool screen and the previously described one is the number of input filenames required. All other options are identical and their description will not be repeated here.



**Generate WAVES Testbench**

The connector filename

The lookup table filename

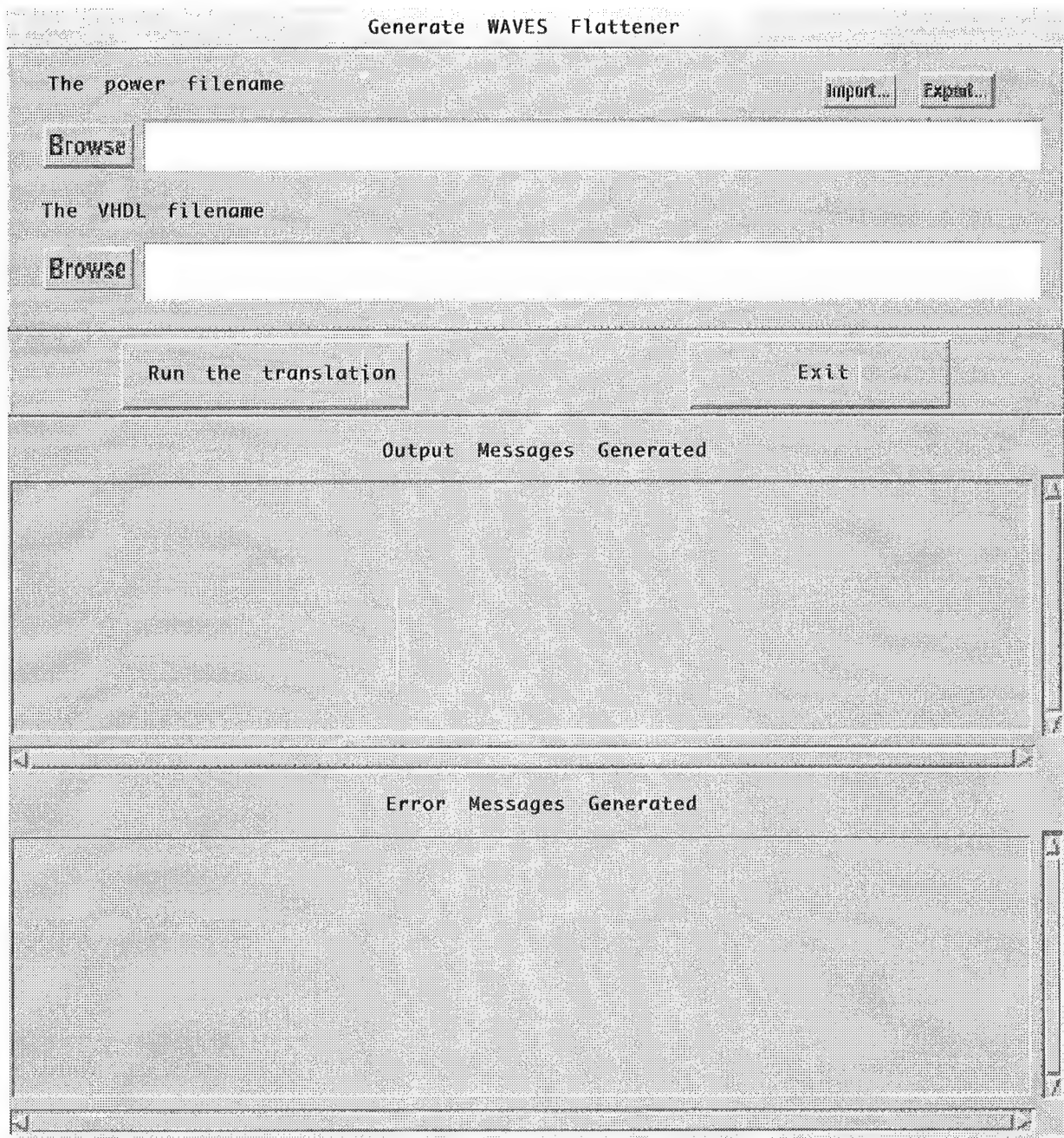
The power filename

The VHDL filename

**Output Messages Generated**

**Error Messages Generated**

**FIGURE 3.8. GENERATE WAVES TESTBENCH INPUT SCREEN**



**FIGURE 3.9. GENERATE WAVES FLATTENER INPUT SCREEN**

### 3.2.1.3 Generate Partial Lookup Table

The initial tool described, Generate WAVES Testbench, requires a lookup table as one of its inputs. The lookup table is a list of the generic component signal names and pin numbers of the actual physical components. The pin direction, either input, output or bi-directional, is also included in the lookup table. This information is needed to produce a netlist used by the ATE for eventual diagnostic purposes. The netlist is an output of the 'Generate WAVES Testbench' tool. The

Generate Partial Lookup Table tool will create most of the lookup table automatically from the information contained in the VHDL model. Everything except pin numbers are created (see Section 1.1.1 in Appendix B). The only input required to execute this tool is the VHDL structural filename and the name of the output file where the partial lookup table will reside. The initial input screen is presented in Figure 3.10. Note that the user will need to edit the partial lookup table prior to using it in the "Generate WAVES Testbench" tool.

Generate partial lookup table

The new table name(editable)

The VHDL filename

Output Messages Generated

Error Messages Generated

FIGURE 3.10: GENERATE PARTIAL LOOKUP TABLE INPUT SCREEN



### 3.2.2 Vector Translation Tools

By selecting the Vector Translation Tools option from the main menu screen the following options are presented:

- Generate Vector Database
- GR2751 Translator (TGO)
- Generate TDL

Generate Vector Database takes the output of the VHDL simulation (using the WAVES testbench created with the 'Generate WAVES Testbench' tool), and creates files that are needed by the GenRad 2751 Genisys™ tool for TPS development. This tool also creates a TRF file for translation into a TDL file, and an input file used by the 'GR2751 Translator (TGO)' tool. Refer to section 2.4 of this guide for additional information on all vector translation tools.

The GR2751 Translator (TGO) tool creates a vector file in the GenRad 2751 ATE format known as TGO. The file is used for functional testing. Finally, the Generate TDL tool takes the TRF file and produces a TDL and PINS file needed by the IN-STEP program for creating tester specific test files.

#### 3.2.2.1 Generate Vector Database

The input screen for this tool is provided in Figure 3.11. Note that this screen looks exactly like all other tool screens. The first input required is the name of the output file created by the VHDL/WAVES testbench simulation. If the Browse function is selected, a filename with the '.tb' extension is searched for. This is reflected in Figure 3.12. Upon selection of the <filename>.tb file, the ID Workbench will automatically select the signal filename, <filename>.sigs, and place it on the signal filename line shown in Figure 3.11. This assumes that the .sigs filename is the same as that used for .tb. If this is not the case, the user must find the correct <filename>.sigs file using the Browse function, or type it in manually. Note that if the user types the name in manually for any input file, the complete directory string must be entered. For example,  
/export/home2/elefante/TOOLS\_GUI/INPUTS/vidal.sigs.

Generate Vector Database

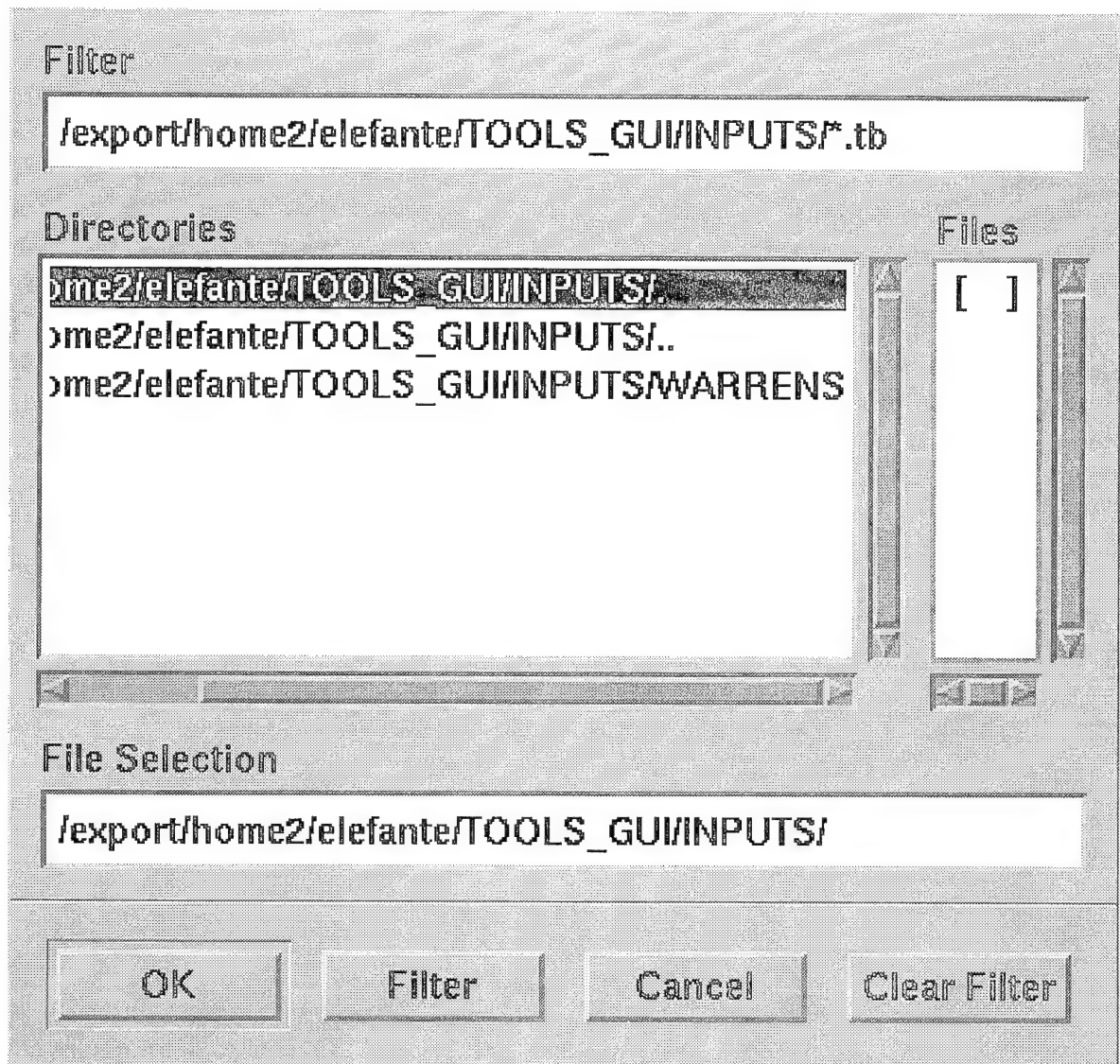
The output of the testbench simulation

The signal filename

Output Messages Generated

Error Messages Generated

FIGURE 3.11. GENERATE VECTOR DATABASE INPUT SCREEN

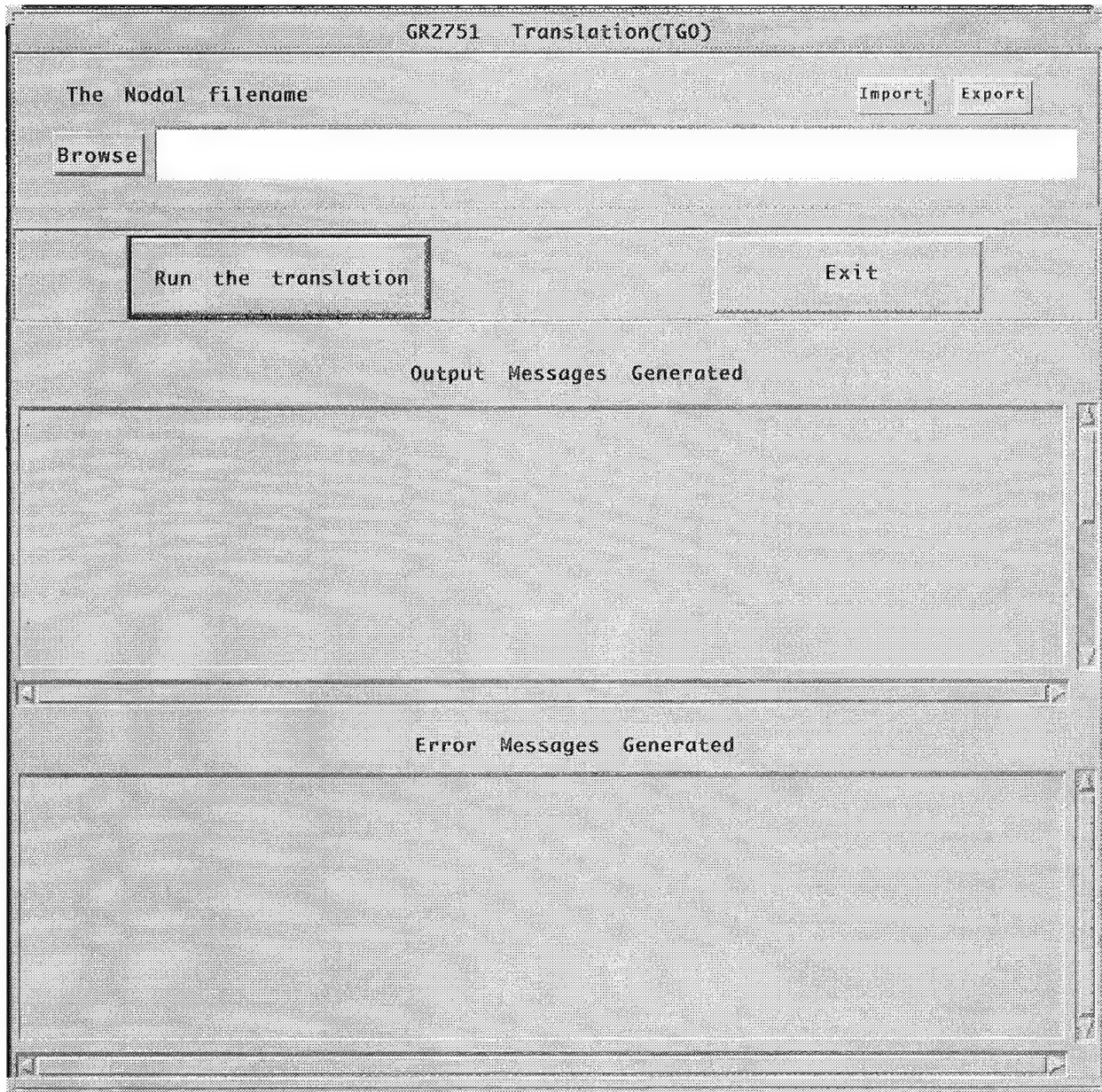


**FIGURE 3.12. SELECTING THE OUTPUT OF THE TESTBENCH SIMULATION  
FILENAME**

#### 3.2.2.2 GR2751 Translator (TGO)

This tool is described in Section 2.4, and in Appendix B. The initial input screen is provided in Figure 3.13. The Nodes filename is the output of the Generate Vector Database tool described in the Section 3.2.2.1, and will be in the format <filename.nodes>. After selecting the file, select 'Run the translation' to execute the tool and generate a .TGO file, which will be used by the Genisys™ tools to create a GenRad 2751 TPS.





**FIGURE 3.13. INITIAL INPUT SCREEN FOR GR2751 TRANSLATOR (TGO) TOOL**

### 3.2.2.3 Generate TDL

This tool is described in section 2.3. The input is the <filename>.trf created by the Generate Vector Database tool described in Section 3.2.2.1. The initial input screen is provided in Figure 3.14. As before, once the correct input file is selected, click on 'Run the translation' to execute the tool and create a TDL description file for the UUT.

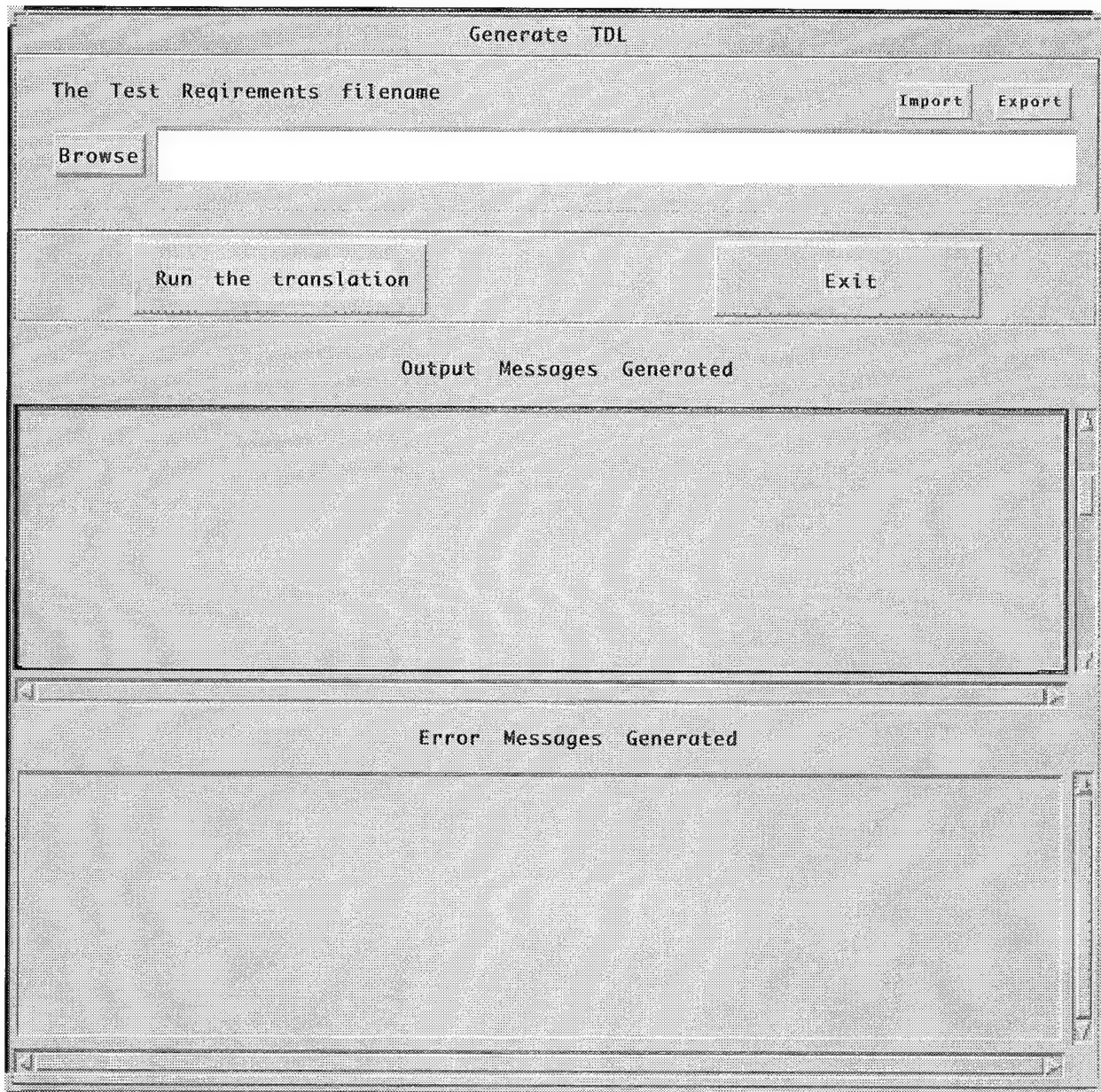


FIGURE 3.14: INITIAL INPUT SCREEN FOR GENERATE TDL TOOL

## 4.0 AN EXAMPLE

The purpose of this section will be to step through an example that will exercise each of the primary tools. The example to be used is of a sample circuit that was developed to test the tools in the ID Workbench during the TISSS Enhancement, Awareness and Management (TEAM) program. The VHDL filename is 'vidal.vhd'. The circuit schematic, along with the VHDL file description is provided in the Appendix C. Other input files, as well as files created by the individual tools, are presented within this section. To obtain a copy of the 'vidal' circuit files that are described herein, contact Mr. Willis Horth or Mr. James Nagy of RL/ERDD via telecon at (315) 330-2241 or DSN 587-2241. Either individual can also be reached via email. The current addresses are:

Willis Horth: horthw@rl.af.mil  
James Nagy: nagyj@rl.af.mil

### 4.1 Generating a Board-Level TPS For The GenRad 2751

The steps involved in using the ID Workbench to aid in TPS development are outlined below. In general, to execute the Test Automation Toolset, an assumption is made here that a VHDL model and all other input files (e.g., .CON, .POW, .TBL) in the format described in Appendix B and reference [3] exists. It is also assumed that a WAVES dataset describing the input stimulus, as a minimum, exists in the format described in references [3] and [4] and that the user has a VHDL simulator compatible with IEEE Std 1076.

- Step 1 (optional): Create a partial lookup table by executing the 'Create Partial Lookup Table' tool
- Step 1a (optional): Create .TBL file from partial lookup table file
- Step 2: Execute the 'Generate WAVES Testbench' Tool
- Step 3: Simulate the resulting WAVES Testbench using the appropriate WAVES Dataset.
- Step 4: Execute the 'Generate Vector Database' Tool
- Step 5: Execute the 'GR2751 Translator' Tool
- Step 6: Execute the 'Generate TDL' Tool
- Step 7: Execute the IN-STEP tool using output of step 6 and GenRad 2751 specific template files (refer to Figure 1.1).

In addition to contacting the above listed individuals for a copy of the ID Workbench tool, the same individuals should be contacted for a copy of the IN-STEP tool (including the IN-STEP users guide) and the GR2751 specific template files.

#### 4.1.1 Step 1 (optional): Create a partial lookup table by executing the 'Create Partial Lookup Table' tool

Assuming that the user has started the ID Workbench and is at the main menu screen (see Figure 3.0), use the mouse to choose Test Automation Tools and highlight the Generate Partial Lookup Table option. The Generate Partial Lookup Table screen requires two entries, the new table name, and the VHDL filename. Using the Browse button, find the directory where the VHDL file "vidal.vhd" resides, and choose the file "vidal.tbl" for the new table name. Once selected, choose OK, and return to the option screen. Next, edit the filename chosen by clicking on the new table name line and then use the keyboard to delete the old name and enter the name "new\_lookup\_table.tbl". Refer to Figure 4.1 to see what this should look like. Note that the directory structure may not be the same as on your own system. Next, use the browse button again to choose the VHDL filename vidal.vhd. Once again, choose OK to return to the main screen. Once this step is complete, click on the "Run the translation" button. The output shown in the Output Messages Generated box shown in Figure 4.1 should appear. If any error messages appear in the Error Messages Generated box, determine what the problem is from the error messages, make any required changes, and repeat the process. Should any problems still exist, contact Mr. James Nagy at the phone number provided earlier in this section.

Given that the partial lookup table was successfully created, a small window will appear with the output files generated, as shown in Figure 4.2. If desired, this file can then be viewed and/or edited using the systems text editor. Since this is a partial lookup table, the file will have to be edited before it can be used in the next step. Therefore, double click on the filename shown in Figure 4.2. Figure 4.3 shows what the beginning of this file looks like. From within the text editor, the pin numbers for each of the devices listed in new\_lookup\_table.tbl can be filled in. The completed lookup table is shown in Appendix B, under the Lookup Table (.tbl) subsection to Section 1.1.1 INPUTS. It is not necessary to modify this file, however, as the file "vidal.tbl" contains the information shown in Appendix B. Completion of the table is left to the user as an exercise. After exiting the text editor, click on the word "EXIT" shown in Figure 4.2 to return to the main tool screen.

#### 4.1.2 Step 2: Execute the 'Generate WAVES Testbench' Tool

The next step is to generate a WAVES testbench by executing the Generate WAVES Testbench option. Return to the main menu screen and once again choose the Test Automation Tools option and then highlight the Generate WAVES Testbench option. The option screen shown in Figure 3.1 will appear. Use the browse buttons to choose the necessary files and then "Run the translation". Figure 4.4 shows how the screen should look with the correct filenames and Output Messages Generated box. Note that each of the required filenames have different extensions but the same name, "vidal". As before, if error messages are generated, determine the cause, make corrections, and re-run this option.



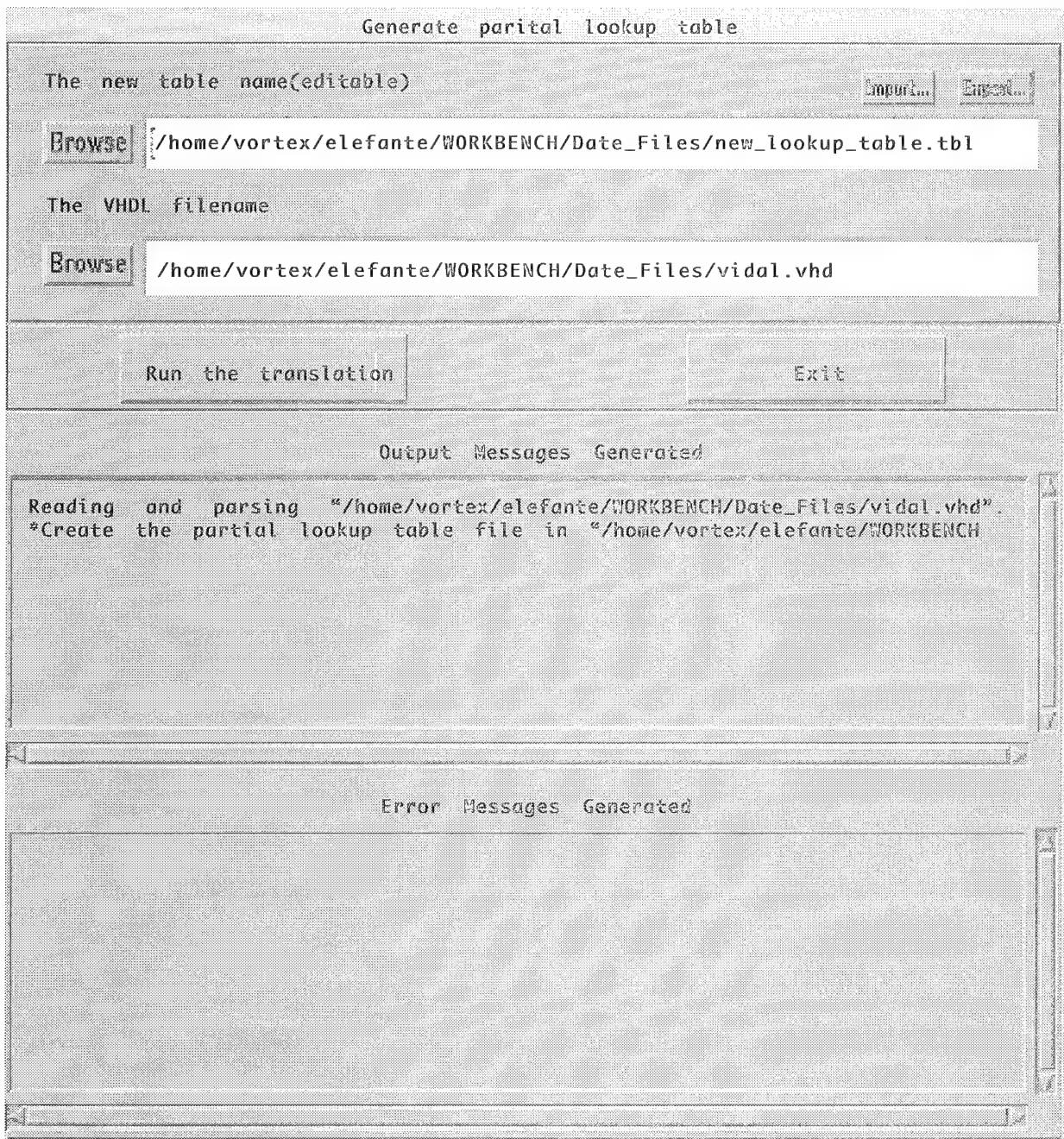
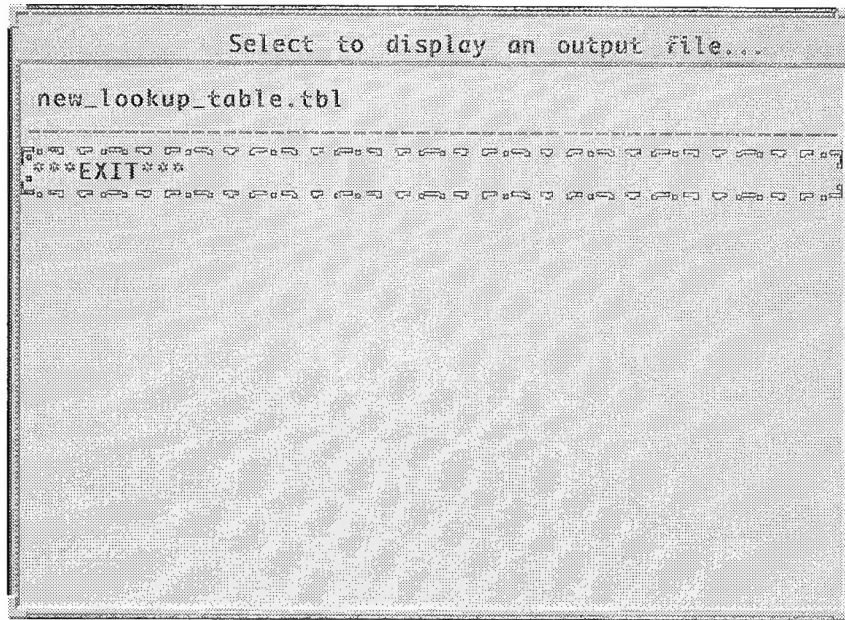


FIGURE 4.1: PARTIAL LOOKUP TABLE SCREEN

A successful run will once again result in the small window showing the output files generated. For the example, Figure 4.5 shows what this will look like. Once again, double click on any of the filenames listed to view or edit a file. After exiting the text editor, click on the word "EXIT" shown in Figure 4.5 to return to the main tool screen.





**FIGURE 4.2: OUTPUT FILE SELECTION SCREEN**

4.1.3 Step 3: Simulate the resulting WAVES Testbench using the appropriate WAVES Dataset.

The next step would be to simulate the resulting testbench created in step 2, using an appropriate WAVES dataset and external vector file. This step has already been completed for this example and the output files are part of the example files provided. Specifically, the file "vidal.tb" is the resulting output of the VHDL simulation.

4.1.4 Step 4: Execute the 'Generate Vector Database' Tool

The next step in this process is to create the GenRad diagnostic probe database "vidal.cap", and a performance vector file in a format that can be translated into the GenRad ".TGO" format required by the Genisys™ tools. The ".cap" file is directly useable by the Genisys™ Tools in creating a GR2751 TPS.

This step is accomplished by running the Generate Vector Database option from within the Vector Translation Tools option found in the main menu screen of the ID Workbench. Therefore, return to the main menu, and select Vector Translation Tools. Next, highlight the Generate Vector Database option. The screen shown in Figure 3.11 will appear. Use the browse buttons to choose the input files vidal.tb and vidal.sigs. Next, select the Run the translation button. Figure 4.6 shows the resulting screen. The output files that should also appear in the smaller window (see Figure 4.5) are vidal.cap and vidal.nodes.

```
Text Editor V3.4 [talon] - new_lookup_table.tbl, dir: /home/vortex/elefante/WORKBENCH/Date_Files/new_lookup_table.tbl
File View Edit Find
-- Filename : /home/cortex/elefante/WORKBENCH/Date_Files/new_lookup_table.tbl
-- Created : 28-Aug-1995 16:32:08
--
-- Created by: T2-1 Version 1.36
--
-- IIT Research Institute -- Copyright August 1994
-- 201 Mill Street
-- Rome, NY 13440
-- (315) 339-7119
-- runkle@mail.iitri.com
-- wswavely@mail.iitri.com
-- jbeaton@mail.iitri.com
--
-- The lookup table has the following format...
--
-- <Component Name>
-- <formal pin name> <pin number> <mode>
-- ...
--
-- T2-1 designates a "#" to represent the undefined pin number
-- that the engineer must determine and enter.
-- T2-1 designates a "?" to represent any undefined mode (direction)
-- that could not be specifically determined from the model.
-- T2-1 designates "****TYPE****" to represent the component type
-- that the engineer must determine and enter.
-- Legal modes are:
-- I or i for input
-- O or o for output
-- B or b for bi-directional
-- 1 for on/power
-- 0 for off/ground
-- - for nothing other than a place holder
-- (this usually accompanies a no-connect pin
-- which is also designated by a '-')
--
--TTL00 ****TYPE**** --COMPONENT & TYPE
--A1 # i
--A2 # i
--A3 # i
--A4 # i
--B1 # i
--B2 # i
--B3 # i
```

FIGURE 4.3: TEXT EDITOR SCREEN

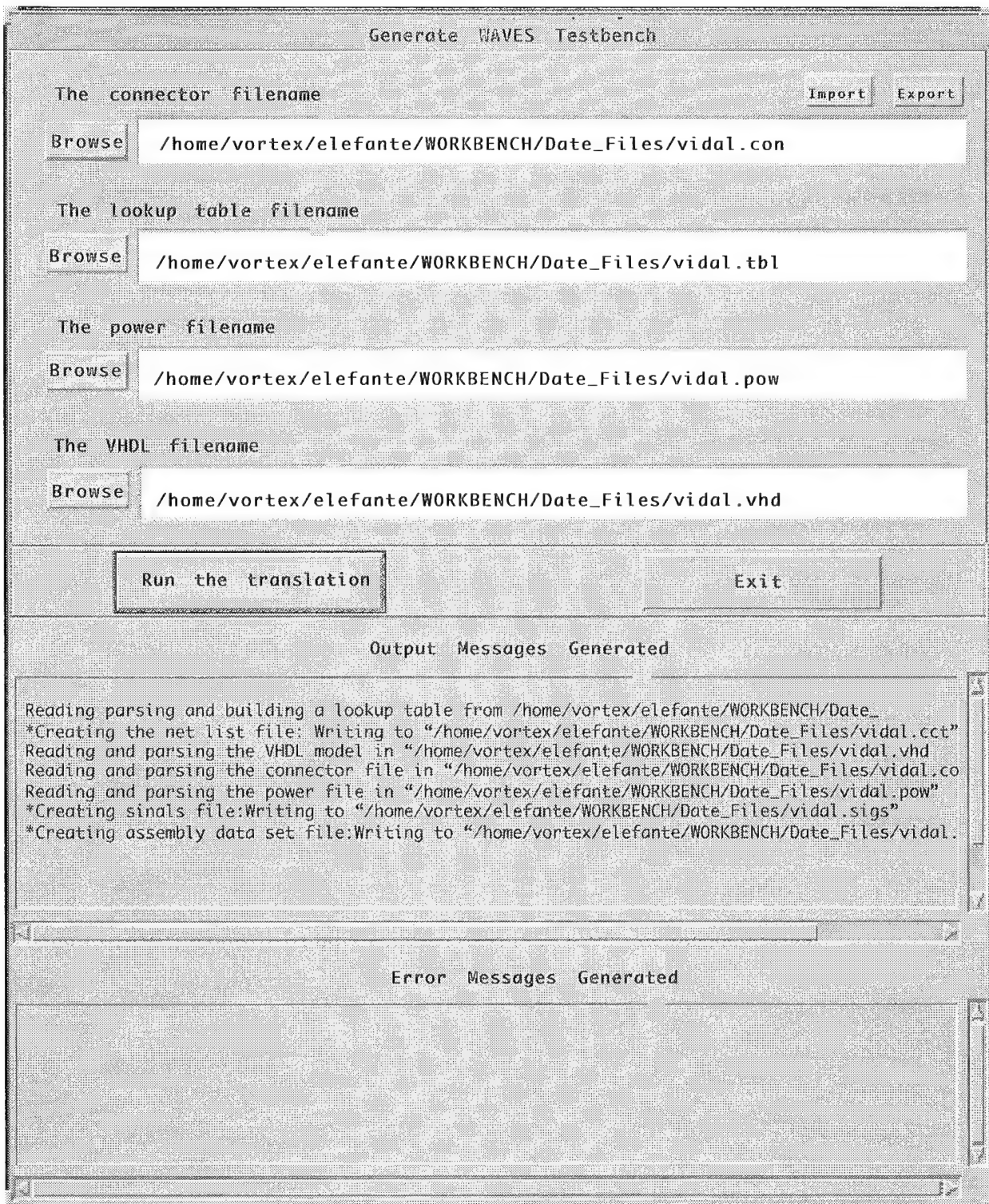


FIGURE 4.4: WAVES TESTBENCH GENERATOR SCREEN

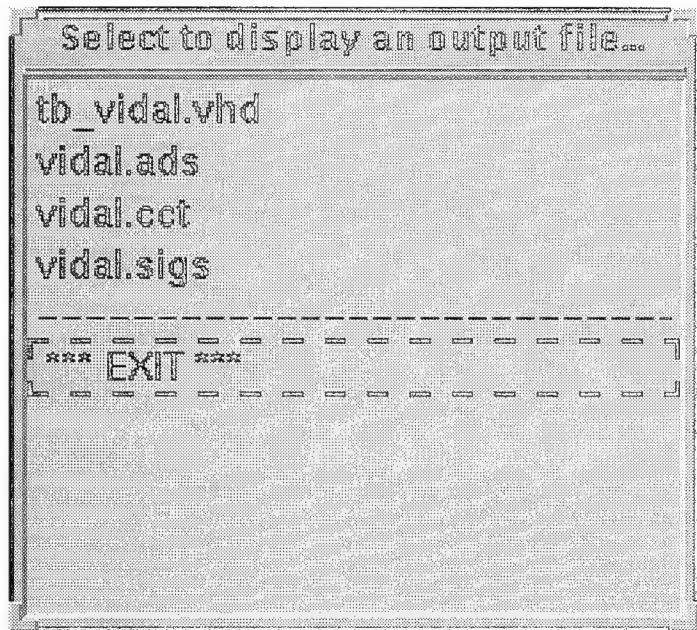


FIGURE 4.5 WAVES TESTBENCH OUTPUT FILES

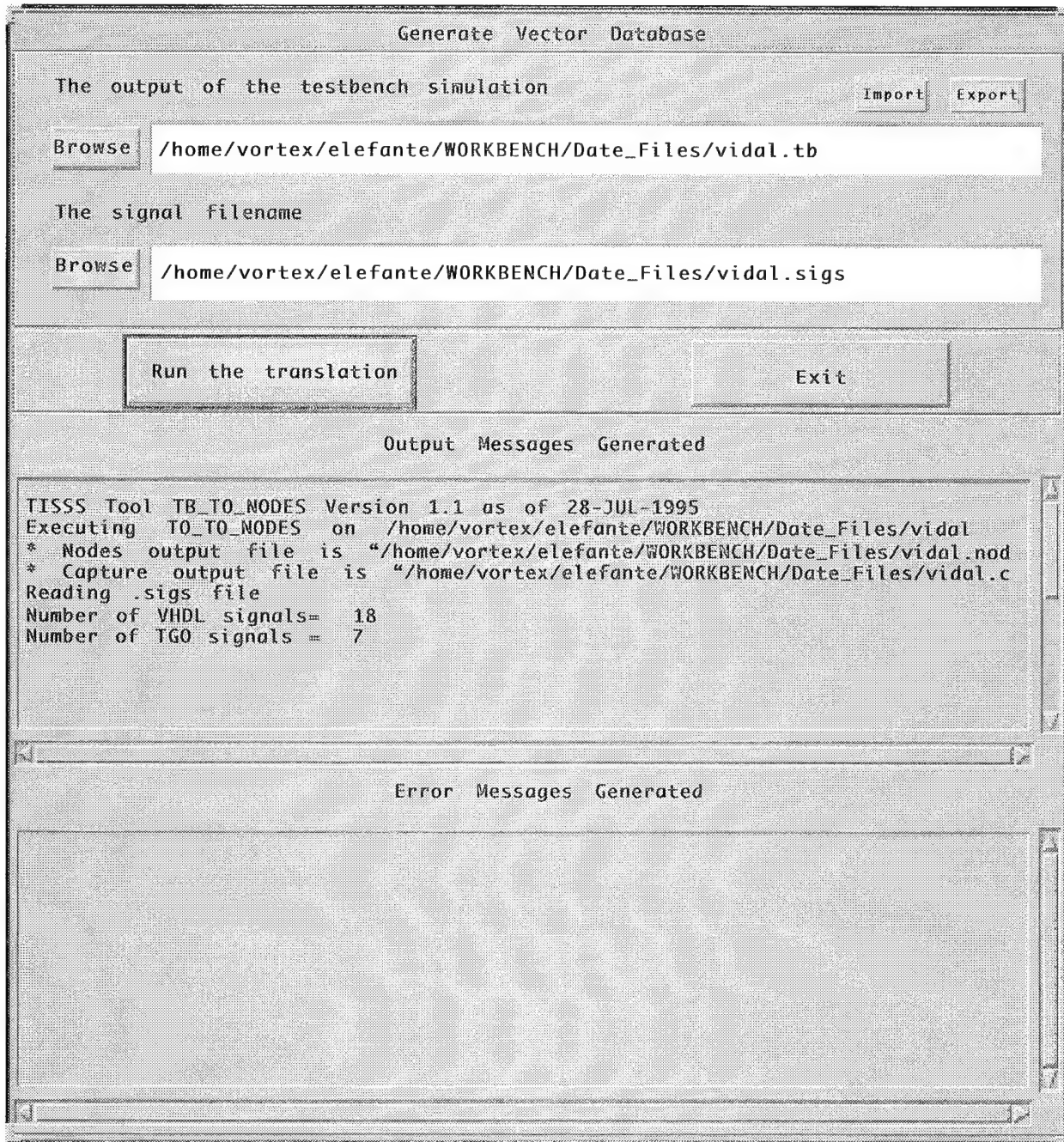
#### 4.1.5 Step 5: Execute the 'GR2751 Translator' Tool

As explained in Step 4 above, the performance vector file must be translated into the .TGO format before running the Genisys™ tools. This is accomplished by executing the GR2751 Translator (TGO) option, also found in the Vector Translation tools option in the main ID Workbench menu screen. Once this option is selected, the screen shown in Figure 3.13 will appear. Once again, use the browse button to choose the filename required; in this case, *vidal.nodes*. Once the file is selected, Run the translation. Figure 4.7 shows what the screen should look like, while Figure 4.8 shows the output files generated. Note that in addition to the .tgo file, a file called *vidal.ctx* is also generated. The .ctx file is an additional file required by the Genisys™ tools for development of a probing database.

#### 4.1.6 Step 6: Execute the 'Generate TDL' Tool

In addition to the vector information needed to test a UUT on an ATE, other files are required for controlling the ATE, as well as for performing other tests, such as power, current and parametric testing. The TDL files, generated automatically by the Generate TDL tool are designed to capture such test requirements for the board level. The TDL files are then used as input to the IN-STEP tool, along with tester specific template files, to generate the remaining information needed to develop a TPS. To generate the TDL files, choose the Generate TDL option from within the Vector Translation Tools option of the main menu screen. The screen shown in Figure 3.14 will appear.





**FIGURE 4.6: GENERATE VECTOR DATABASE SCREEN**

Use the browse button to select the Test Requirements filename (TRF) `vidal.trf`, which was provided with the example files. If a TRF file had not existed previously, one would have been created when the Generate Vector Database option was executed. Since this file already existed, the file was not created. Once the `vidal.trf` file is selected, run the translation. The screen in Figure 4.9 shows the resulting screen. The resulting output file is `vidal.tdl`.



4.1.7 Step 7: Execute the IN-STEP tool using output of step 6 and GenRad 2751 specific template files (refer to Figure 1.1).

The last step to creating the files needed for a GR2751 TPS is to execute the IN-STEP tool using the GR2751 specific template files and the TDL file created in Step 6. Because the IN-STEP is not yet part of the ID Workbench, it is not described here. As noted, a copy of the IN-STEP tool, user guide, and ATE specific template files can be obtained from RL. For further information on IN-STEP, see reference [5 ].

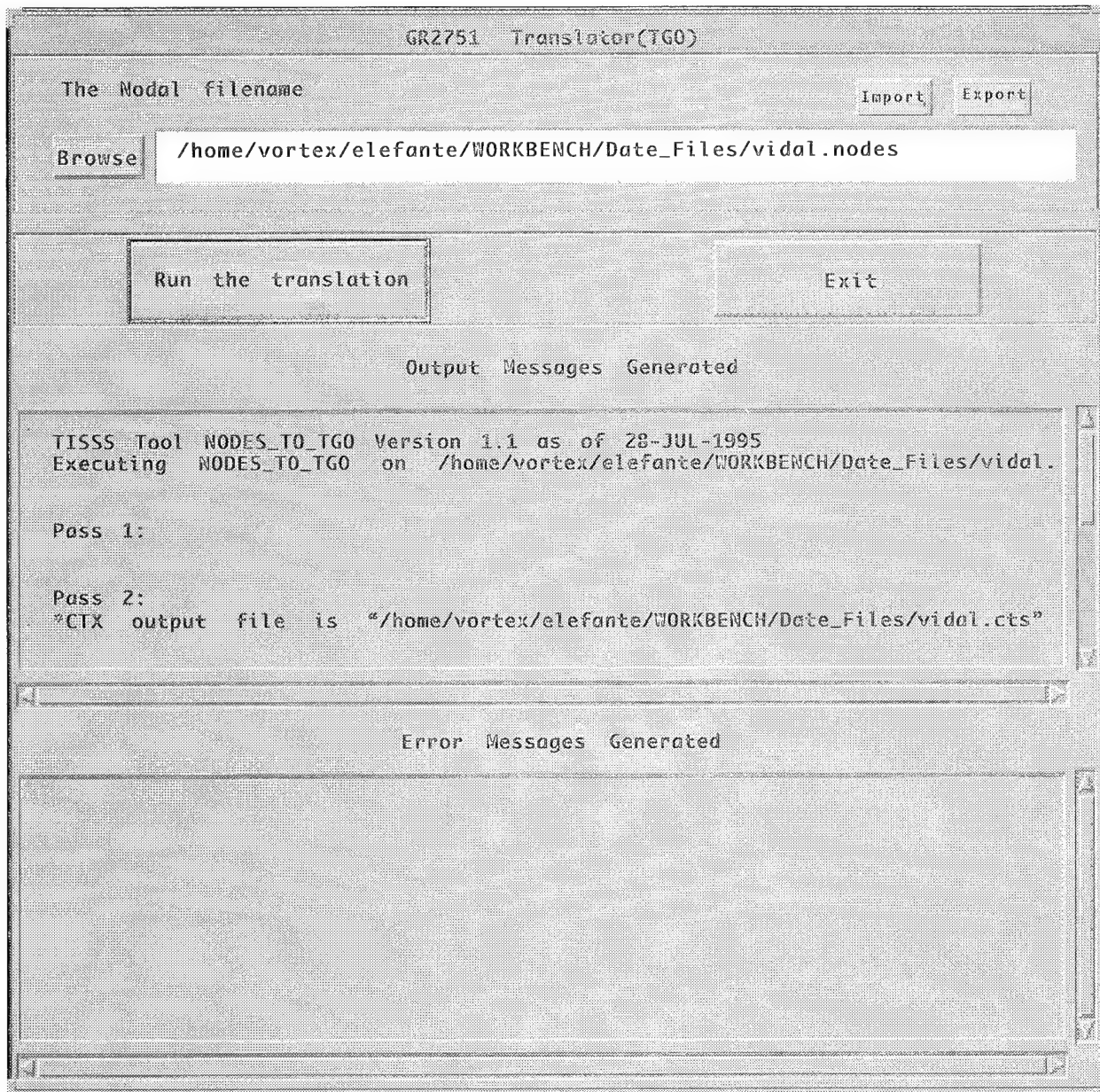


FIGURE 4.7: GR2751 TRANSLATOR SCREEN

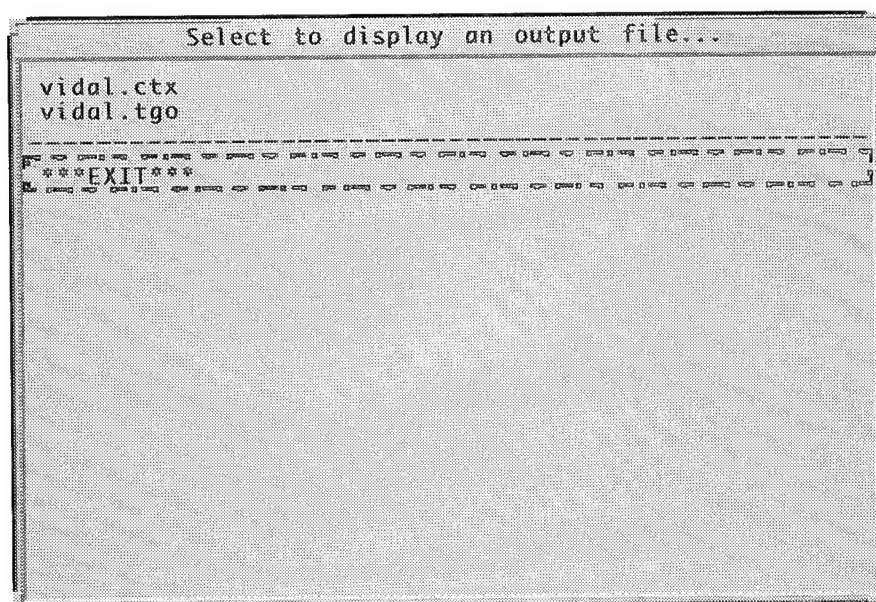


FIGURE 4.8: VECTOR TRANSLATION OUTPUT FILES

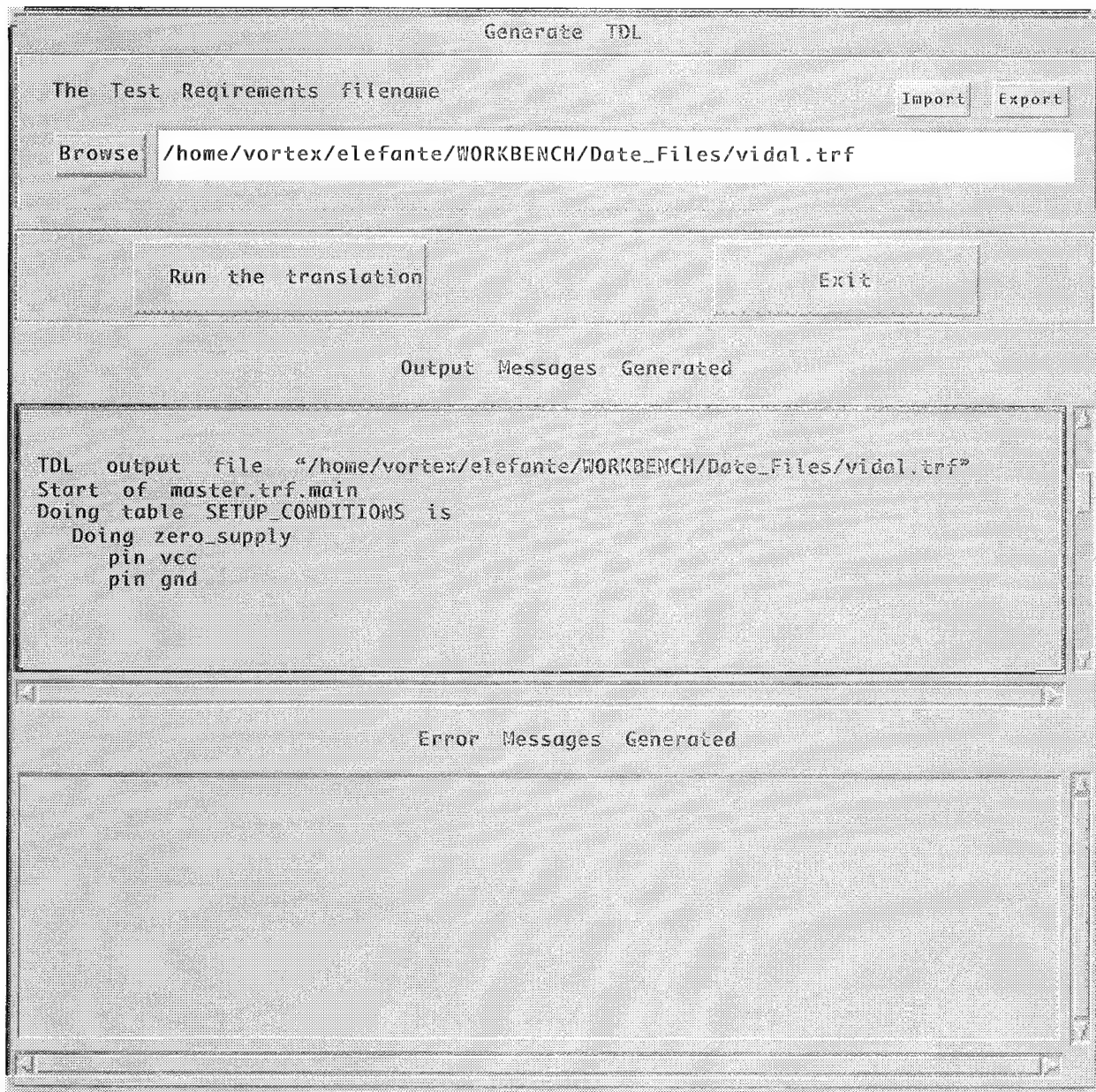


FIGURE 4.9: GENERATE TDL SCREEN

## REFERENCES

- [1] Debany, Warren H., et. al., "CONVERTING TEST REQUIREMENTS INTO TEST PROGRAM SETS," Proceedings, AUTOTESTCON '93
- [2] Nagy, James M., and Newbert, Jeffery, "Capturing Board-Level Test Requirements in Generic Formats," proceedings, AUTOTESTCON '94
- [3] Debany, Warren H., et. al., "An Update to Applications of Open Standards to Test Automation for Board Level Testing," proceedings, AUTOTESTCON '94
- [4] Beaton, Joseph, et. al., "A Generic VHDL Testbench to Aid in Development of Board-Level Test Programs," proceedings, AUTOTESTCON '94
- [5] Hanna, James P., and Horth, Willis J., "A NEW METHODOLOGY FOR TEST PROGRAM SET GENERATION AND RE-HOSTING," proceedings, AUTOTESTCON '91
- [6] Tapscott, M, "GEM, Aftermarket Provide Affordable Solutions as Old Ics Get Scarce," Defense Electronics, October 1993
- [7] Appendix A, ADSF Specification, GenRad Inc.

## APPENDIX A TESTER INDEPENDENT SUPPORT SOFTWARE SYSTEM (TISSS)

### BACKGROUND

#### 1. INTRODUCTION

##### 1.1 Purpose

The purpose of Appendixes A-C is to present additional information on the Tester Independent Support Software System (TISSS) Toolset (T<sup>2</sup>). Specifically, Appendix A provides background information on the TISSS program, while Appendixes B and C provide more detail on the following ID Workbench Tools or Options:

- WAVES Testbench Writer
- Generate Partial Lookup Table
- Generate Vector Database
- GR2751 Translator (TGO)

The T<sup>2</sup> is currently designed to automate the process of developing test files that can be used to generate Test Program Set (TPS) software required to test a digital circuit board on either the GenRad 2751 or MATE-390 tester. These appendixes will provide background on the TISSS program, both past, current and future, as well as information on the specific T<sup>2</sup> input file requirements and output file formats.

The T<sup>2</sup> is based on developing TPS software via simulation in the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL). However, parts of the T<sup>2</sup> could be used to re-host current TPS software, provided all files are in the proper formats as described herein.

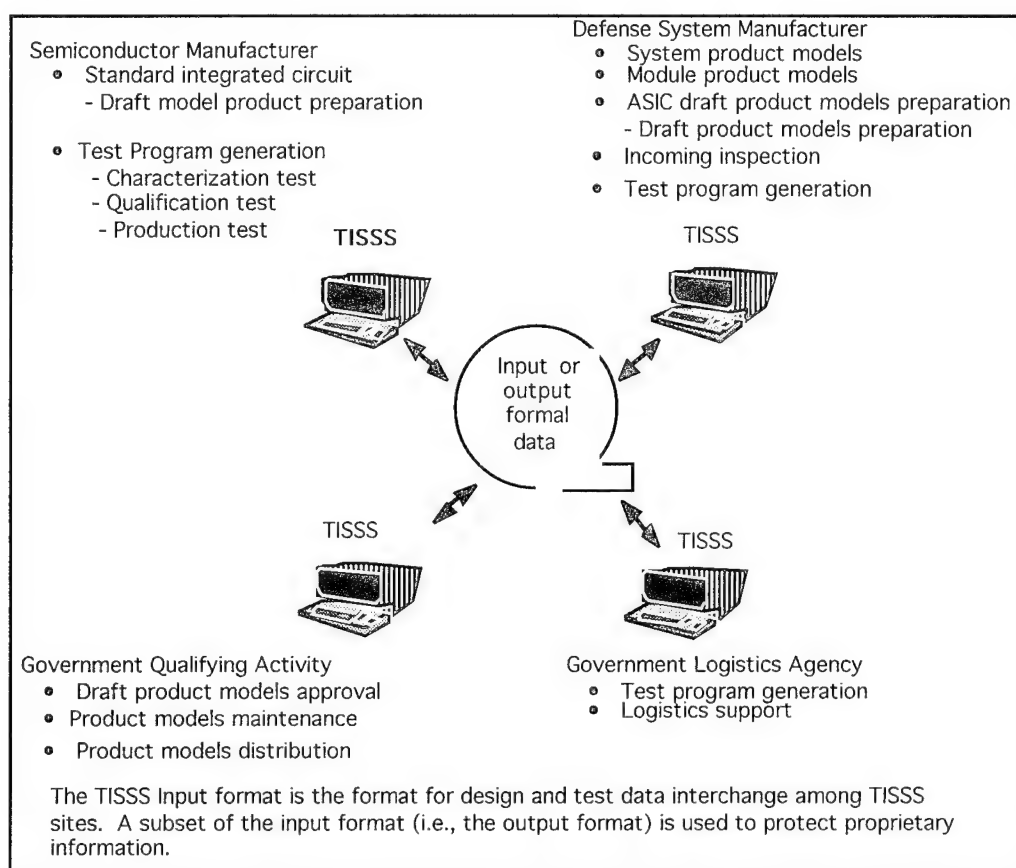
##### 1.2 TISSS

###### 1.2.1 Past

Before describing the specifics of the TISSS toolset, it is appropriate to provide some background information on TISSS. TISSS was originally developed in the mid-1980's time frame as a means to electronically capture test requirements information for microcircuits embodied in MIL-M-38510, "General Specification for Microcircuits " slash sheets and/or manufacturer's data sheets. The information was captured in a data format defined as Test Description Language (TDL). In addition to information such as voltage and current levels, other component characteristic information such as test



propagation delay times, test temperature levels (needed for qualification testing), and test philosophy information (based on technology type (e.g., CMOS) ) is also part of the TISSS database structure. The original intent of TISSS was to enable the exchange of this information among semiconductor manufacturers, defense system manufacturers (sometimes referred to as Original Equipment Manufacturers (OEMs)), government qualifying activities, and government logistics agencies. An example of this exchange system is provided in Figure A1.



**FIGURE A1: EXAMPLE OF A TISSS ENVIRONMENT**

The major advantage to TISSS is the concept of capturing test requirement information in a format that is non-proprietary and independent of any Computer-Aided Design (CAD) system or Automatic Test System (ATE). The advantage of this is illustrated in Figure A1. In this example "TISSS system", a semiconductor manufacturer may use the system to capture design and test information and to generate a draft product model. The data may then be submitted to the government qualifying activity for product model and data approval. A defense system manufacturer or integrator may use the system to qualify complex application-specific devices or to query the system (at the government qualifying activity) for design, test, and product model information system. The government logistics agencies may use the TISSS to generate test programs for the product model. The capability to perform each of the described tasks lies in the fact that the TISSS data is in a standardized format that is recognizable by those who wish to

access the data. If one knows the format of this data, they have the means to translate the information to make it compatible with other processes, such as development of a test program. Herein lies the power in the TISSS concept of non-proprietary data capture.

In addition to test requirements information captured in TDL, a means to capture test vector information in a standardized format and a post-processor program was also developed under the original TISSS effort. The test vector format was eventually replaced by the Waveform And Vector Exchange Specification (WAVES) format for describing simulation and test vectors. WAVES is IEEE standard 1029.1, and is a subset language of VHDL (IEEE standard 1076). The post-processor, named the INdustry Shared TEST Processor (IN-STEP), is designed to read TDL files, device pin information files, test philosophy files and test vector information and, using a series of ATE specific template files, create a test program that can be compiled and run on the target ATE. A flow diagram showing an example of how IN-STEP can be used is shown in Figure A2.

### 1.2.2 Present

Since the original TISSS program was completed in 1986, TISSS, and its meaning, has gone through several changes. One change was that the microcircuit database portion of TISSS was not something that generated a great deal of interest. The primary reason was that MIL-M-38510 was eliminated as a government standard and replaced by the Quality Manufactures List (QML) procedures and standards. Therefore, the need to have such information in this format, essentially went away. Another change in TISSS was that ANY standard for electronics (e.g., design, test, etc.), that captures design and test information in non-proprietary formats was adopted by those who developed TISSS at RL under the "TISSS umbrella". This included the standard on VHDL. Also, since the completion of IN-STEP, the process of going from TDL and WAVES out to a tester and actually testing a device had been demonstrated for a number of microcircuits. Much of the demonstration was done in-house at RL, and includes the following devices:

- an LRM from the F-22 Advanced Tactical Fighter program
- the "Pathfinder" chip (a 181-lead package with 141 I/O pins)
- a RICMOSIV "Evaluation Metal" chip (a 256-lead package with 210 I/O pins) delivered under RL's Radiation-Hardened 32-Bit (RH32) Processor Program
- a 54LS161 synchronous 4-bit counter that was part of RL's Field Failure Return Program.

All devices were tested on a Teradyne J953 components tester resident at RL.

Figure A3 shows the process by which test programs were created for the listed devices. In each case, test vectors were either developed from device specifications or, in the case of the RH32 devices, delivered by a contractor. Other information, such as device voltage and current levels and test propagation delay times were captured in a simple

```

graph LR
    subgraph TDL
        TP[Test Philosophy  
- test class  
- test case  
-test priority]
        PD[Pin Definitions  
- pin name, term position  
- pin groupings]
        TDSL[Test descriptions and device specifications and limitations]
        TC[Tester configuration or device interface board file]
        TI[Timing information from WAVES translator]
    end

    subgraph IN-STEP
        TDP[TDL Parser  
Creates a list containing 15 sublists:  
4 REQ, 3 PIN  
6 TDL]
        TDP2[Test Data Parser  
- Creates individual list from free-formatted file]
        EP[Expression parser  
- Creates individual list from file in list format]
    end

    subgraph OUTPUT_DIRECTORY [OUTPUT DIRECTORY]
        PMH[pinmap.h]
        CHM[chann_map.h]
        PLH[pin_lists.h]
        PT[pindef.tem]
    end

    TP --> TDP
    PD --> TDP
    TDSL --> TDP
    TC --> TDP2
    TI --> EP

    TDP --> PMH
    TDP --> CHM
    TDP --> PLH
    TDP2 --> PT
    EP --> PT
  
```

The flowchart illustrates the testbench generation process, organized into three main sections: TDL, IN-STEP, and OUTPUT DIRECTORY.

**TDL (Test Data List) Section:**

- Test Philosophy** (includes test class, test case, test priority)
- Pin Definitions** (includes pin name, term position, pin groupings)
- Test descriptions and device specifications and limitations**
- Tester configuration or device interface board file**
- Timing information from WAVES translator**

**IN-STEP Section:**

- TDL Parser**: Creates a list containing 15 sublists: 4 REQ, 3 PIN, 6 TDL.
- Test Data Parser**: Creates individual list from free-formatted file.
- Expression parser**: Creates individual list from file in list format.

**OUTPUT DIRECTORY Section:**

- pinmap.h**
- chann\_map.h**
- pin\_lists.h**
- pindef.tem**

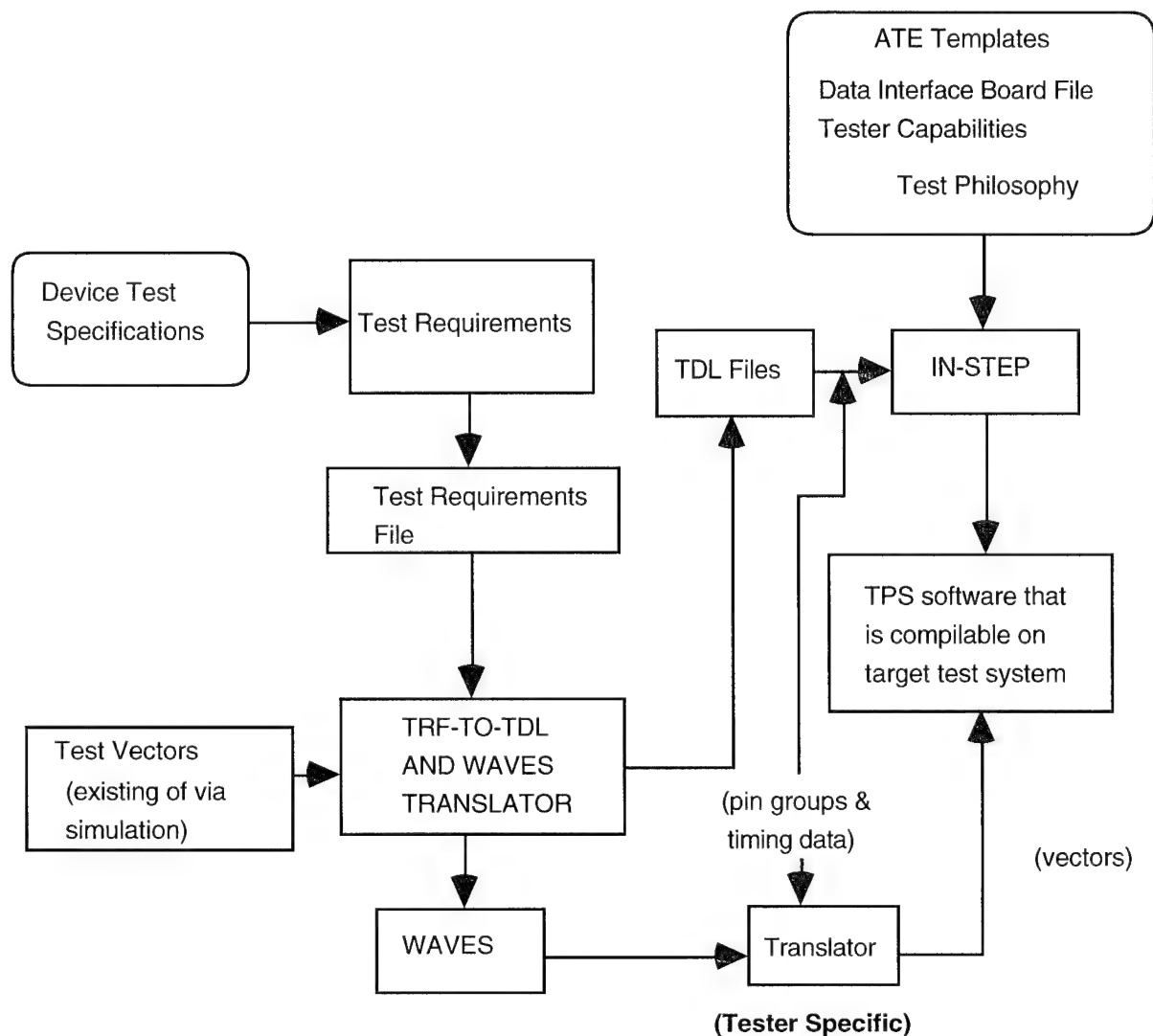
**TEST TEMPLATES Section:**

- pindef.tem**

The process flow is as follows:

- Inputs from the TDL section (Test Philosophy, Pin Definitions, Test descriptions and device specifications and limitations) feed into the TDL Parser.
- Inputs from the TDL section (Tester configuration or device interface board file, Timing information from WAVES translator) feed into the Test Data Parser and Expression parser, respectively.
- The TDL Parser outputs three files to the OUTPUT DIRECTORY: pinmap.h, chann\_map.h, and pin\_lists.h.
- The Test Data Parser and Expression parser output individual lists to the TEST TEMPLATES section, which then feeds into the OUTPUT DIRECTORY.

A-4



### 1.2.3 Future

October 1992, both RL and the RAC assumed a greater role in developing a means to implement the TISSS tools in a board-level test development environment. In conjunction with Science Applications International Corporation (SAIC), who was hired by ADTIC to perform much of the TISSS tool evaluation, TISSS has been expanded to handle board level test. In particular, a test automation path has been developed to go from a board-level design captured in VHDL to test program files that are compatible with a GenRad 2751 tester, and a MATE 390 Atlas tester. The future of TISSS is to improve upon the prototype tools developed for the aforementioned test environments, and to expand the TISSS environment to handle more testers and to improve the TPS development process. The process described by Figure A3 and in reference [1] is currently being used to test an AM2901 chip designed under the Generalized Emulation Microcircuit (GEM) program. More information on GEM can be found in reference [6].



## APPENDIX B

### THE TISSS TOOLSET (T<sup>2</sup>)

#### 1. THE TISSS TOOLSET (T<sup>2</sup>)

Figure B1 presents the current input file requirements and output files produced by the current set of TISSS tools. The following paragraphs will describe all input files and their required formats, the general purpose of each TISSS tool, and the output files produced by the primary T<sup>2</sup> components. Figure B1 presents the process developed to generate GenRad 2751 (GR-2751) compatible test program files. Once created, these files are then used by the GenRad Genisys™ tool to create a test program that can be executed on the GR-2751. Refer to Figure B1 for the information presented in this section. Figure B1 shows five (5) T<sup>2</sup> components that are listed below:

- WAVES Testbench Writer
- Vector Translation Tools
- TRF-TO-TDL Translation
- IN-STEP
- WAVES Vector File Translator

##### 1.1. WAVES Testbench Writer

###### 1.1.1 Inputs

The purpose of this T<sup>2</sup> component is to read in four files as shown in Figure B1: a VHDL file of the board to be tested, a connector (.con) file, a power (.pow) file, and a component look-up table (.tbl) file. The resulting output is a GenRad .ADS file [7], a GenRad .CCT file for each component in the VHDL board model, a VHDL simulation test bench that will be used to simulate the board model, a signals file naming each external signal and its direction (i.e., input, output, bi-directional or power), and a pins file describing each connector pin and its corresponding connector pin number and board signal name. An additional input to the WAVES Testbench Writer is a look-up table containing information on the individual Integrated Circuits (ICs) used on the board to be tested. Part of the look-up table can be created by executing the "Generate Parital Lookup Table" option from within the ID Workbench. All information required in the lookup table, except individual component pin numbers, can automatically be produced from the incoming VHDL structural model. The format for each of these inputs is discussed in the subsections that follow.

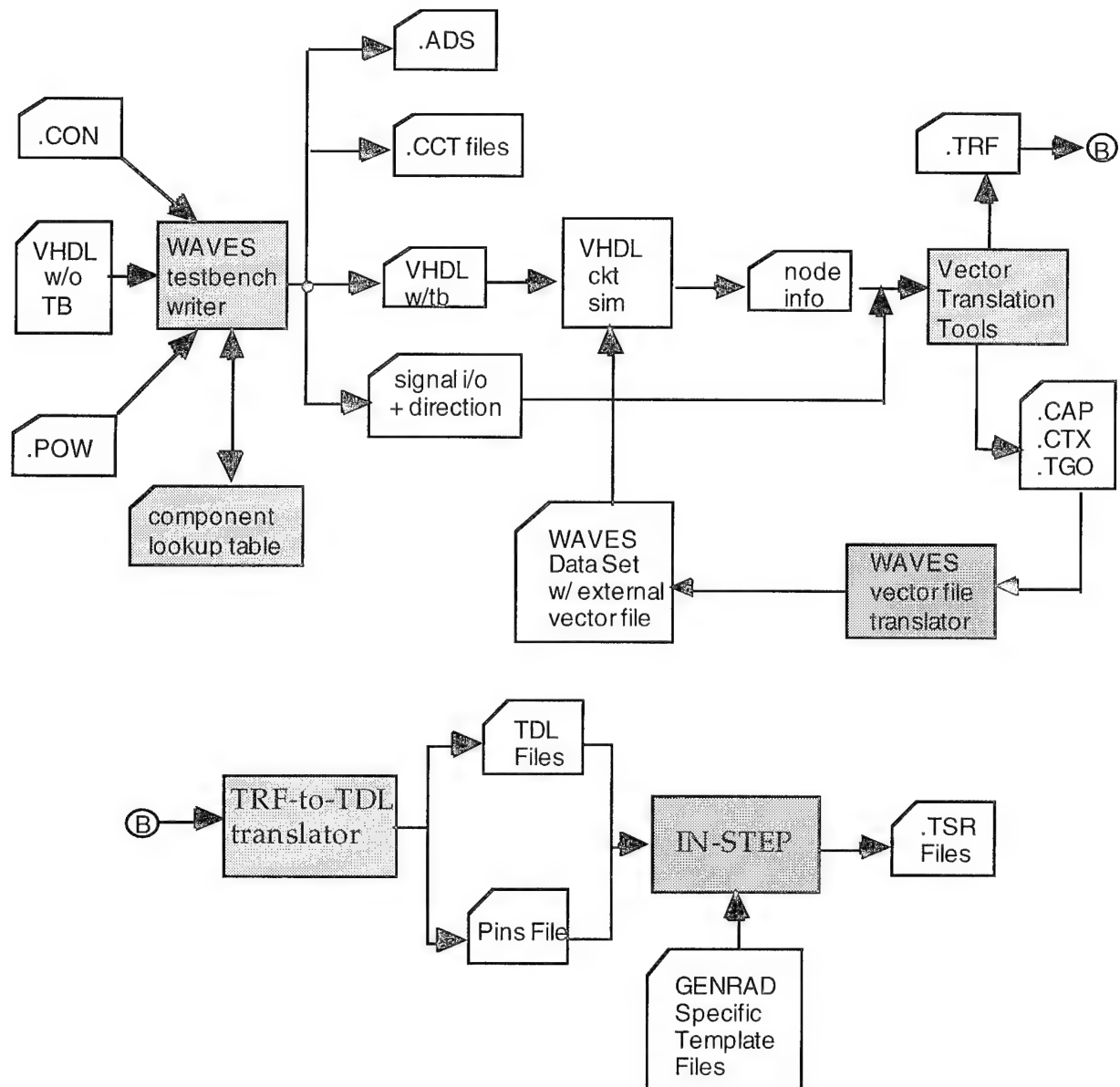


FIGURE B1: T<sup>2</sup> I/O REQUIREMENTS FOR THE GENRAD 2751

### .CON FILE

The information contained in the .CON file is a list of the type of connector (s) used on the board to be tested (herein after referred to as the Unit Under Test (UUT)), and the connector pin names. This information is used along with the information contained in the VHDL structural model and the .POW file to create a GenRad Assembly Data Set (.ADS) file. As described in the .ADS File specification, "The Assembly Data Set File provides a text file interface to the test system assembly database. The file supports descriptions of printed wire boards, discrete components,

and integrated circuits. Assembly data includes information about the assembly, parts, connections, packaging, power, and functional groups. "

The .CON file is a simple text file that contains a header line followed by a list of the connector pin numbers and corresponding pin names. The header line lists the name of the connector (e.g., P1A), followed by the type of connector (e.g., edge\_conn) and the total number of pins on the connector. An example of the header line is provided below.

P1A Edge\_Connector 140

**Note: The header line must be the first non-comment line in the .CON file. Comments are allowed in the .CON file. Comments are identified by the double dash marks (i.e., --) preceding the comment.**

The above header line describes a connector, denoted as P1A, that is an edge connector having 140 pins.

**Note that a space must separate the connector name, connector description, and number of pins. Also, no spaces may exist in the connector type description, and the description is limited to 80 characters.**

Following the header would be a list of those pin numbers that are actually being used on the UUT and the signal name associated with each pin. **The signal names must be the same ones used in the VHDL model.** As an example:

```
1  CONTROL_A
2  DATA_IN
10 DATA_OUT
100 BIT_SIGNAL
```

**The only restrictions on listing the pin number and corresponding UUT signal names are that a space must exist between pin number and signal name, the signal name cannot have any spaces and is limited to 80 characters, and only one pin number/signal name per line.**

An example of a complete connector file for the example circuit shown in Appendix C, Figure C1 is presented below.

```
--
-- VIDAL.CON
-- William G. Swavely
-- 1:56PM 3/10/94
--
```

```

P1A mux_conn1 9
1    P1_1
2    P1_2
3    P1_3
4    P1_4
5    P1_5
6    P1_6
7    P1_7
8    VCC
9    GND

```

### .POW FILE

The information contained in the .POW file is a list of all voltage supply signals used on the UUT, along with the nominal value, maximum current value, the ground signal name that each supply signal is with respect to (WRT), and the start up and shut down delay times associated with each supply signal. A sample .POW file is shown below.

-- NAME	VOLTAGE	MAX. CURRENT	WRT	DELAY
VCC1	5.0V	10A	GND	1S
VCC2	5.0V	10A	GND	2S

The actual .POW file does not have to contain the field headers as shown above. The headers are shown here to explain the requirements of each field and the order in which information is listed in a record. Note that the field header names have a comment symbol "--". This is allowable for those who wish to include the header names in this file. Table B1 below describes each field in the .POW file.

**Note that only voltage supply signals need be listed in the .POW file.**

### VHDL STRUCTURAL FILE

The VHDL structural file is the file that describes the UUT in VHDL. Figure C1 and C1.1, (see Appendix C) will be used to illustrate the format requirements for any VHDL model to be translated by the WAVES Testbench Writer. Figure C1 is a schematic of a simple circuit, and Figure C1.1 is the VHDL description of the circuit.

**Table B1: .POW FILE FIELD DESCRIPTIONS AND RULES**

<u>Field Name</u>	<u>Description</u>	<u>Rules</u>
NAME	voltage signal name	no spaces, maximum of 80 characters
VOLTAGE	The nominal voltage value	enter nominal value in Volts. (i.e., V, mV, etc.)
MAX CURRENT	maximum allowable current for the voltage source	enter value in Amperes
WRT	name of ground source voltage signal is with respect to (wrt)	signal name with no spaces, maximum length of 80 characters
DELAY	The up delay/down delay for power up and power down sequences	enter value in seconds (must have unit of S as shown in example). The value entered will be used for both power up and power down

The VHDL description file shown in Figure C1.1 shows comments in **bold face** that explain the specific rules that must be adhered to for use of the WAVES Testbench Writer component of the TISSS tools. The tool also expects all VHDL model code to be contained in a single file prior to input to the tool. The tool also expects, at most, a single instance of the following blocks of code:

```
BLOCK1:  entity
          ...
          ...
          end
```

```
BLOCK2:  architecture
          ...
          ...
          begin
          ...
          ...
          end
```

```
BLOCK3:  configuration
          ...
          ...
          end
```



The position of each block within the model is not restricted to the order shown above, any legal order is acceptable.

#### LOOKUP TABLE (.TBL)

In addition to developing a VHDL description of the UUT in the above formats, a "lookup" table for each of the individual components contained in the VHDL description file must be defined. The lookup table needs to exist as a single file and component information needs to be in the format shown below.

Component name (as used in the VHDL model)  
 pin name    pin number    pin direction

Note that the component terminal or pin name need not match the signal names connected to a particular device in the VHDL model. For most cases, if the component is a standard device, then the terminal names would be those found in a manufacturer's data sheet for that device. The component look-up table for the components in the sample circuit would look like what is shown below. The information shown in **bold** typeface can be automatically created by the "Generate Partial Lookup Table" tool from the VHDL model information (refer to Section 3.2.1.3). Also note that power pins (VCC and GND in the example presented) do not have a direction, rather they must have a '1' or a '0' for logic level. Since power pins will typically be declared as an input, they will be designated with an **i** in the partial lookup table created by the tool. The user must manually edit the power and ground signals by replacing the **i** with either a '1' or a '0'.

#### **TTL00**

<b>A1</b>	1	<b>i</b>
<b>B1</b>	2	<b>i</b>
<b>Y1</b>	3	<b>o</b>
<b>A2</b>	4	<b>i</b>
<b>B2</b>	5	<b>i</b>
<b>Y2</b>	6	<b>o</b>
<b>A3</b>	9	<b>i</b>
<b>B3</b>	10	<b>i</b>
<b>Y3</b>	8	<b>o</b>
<b>A4</b>	12	<b>i</b>
<b>B4</b>	13	<b>i</b>
<b>Y4</b>	11	<b>o</b>
<b>VCC</b>	14	<b>i -- this must be replaced with a '1'</b>
<b>GND</b>	7	<b>i -- this must be replaced with a '0'</b>

#### **TTL175**

<b>P1</b>	1	<b>i</b>
<b>P2</b>	2	<b>o</b>
<b>P3</b>	3	<b>o</b>
<b>P4</b>	4	<b>i</b>

P5	5	i
P6	6	o
P7	7	o
GND	8	i -- this must be replaced with a '0'
P9	9	i
P10	10	o
P11	11	o
P12	12	i
P13	13	i
P14	14	o
P15	15	o
VCC	16	i -- this must be replaced with a '1'

**NOTE: Device pins that are no connects (NC), MUST be designated by a dash (-) mark in the lookup table. As an example:**

```

TTL02
- 1 - (this is the no-connect pin)
P2 2 I
      •
      •
      •

```

### 1.1.2 Execution

Executing the WAVES Testbench Writer and Partial Lookup Table tools is described in Section 3.0 of the report.

### 1.1.3 Outputs

As depicted in Figure B1 and in the above example, the WAVES Testbench Writer creates four (4) different files as output. Two of these files (.ADS and .CCT) are used directly by the GenRad Genisys™ system. The VHDL w/tb file is the VHDL structural file with a test bench (tb\_vidal.VHD for the example circuit). The testbench file is used to simulate the board model in a VHDL simulator. The signal input/output (i/o) file is used by T<sup>2</sup> component "Vector Translation Tools". The .ADS file was described earlier, and complete information on this file can be found in the GenRad ADSF Specification [7]. The .CCT file contains GenRad .CCT "shell" files in the format shown below, for each component in the board model.

```
IC SN54LS138 (,A,B,C,NG2A, ,NG2B,G1,Y[7],GND, ,
              Y[6],Y[5],Y[4],Y[3], ,Y[2],Y[1],Y[0],VCC)
```

```
INPUT      A B C NG2A NG2B G1 ;
OUTPUT    Y[0:7] ;
SUPPLY1    VCC;
SUPPLY0    GND.
```

IC terminals that are not used(either opens or no-connects), are listed as a space in the above formatted signal list. As an example, the cross reference list for the above 20 pin device is as follows:

**TABLE B2: DEVICE PIN CROSS REFERENCE LIST**

<u>Pin Number</u>	<u>Pin Designation (as listed in the look-up table file)</u>
1	-
2	A
3	B
4	C
5	NG2A
6	-
7	NG2B
8	G1
9	Y[7]
10	GND
11	-
12	Y[6]
13	Y[5]
14	Y[4]
15	Y[3]
16	-
17	Y[2]
18	Y[1]
19	Y[0]
20	VCC

An IC listing is created for every component used in the board design.

The signals (".sigs") i/o file contains a listing of external and internal board signals declared in the VHDL file, including signal direction information and a cross reference table for the external signals that maps the signal name to the connector pin name.

The very first line of the file contains the comment characters '--' immediately followed by two unsigned long words, number of external signals and the number of internal

signals. So, the number of external signals can be found at byte 2 (using C indexing starting at 0) and the number of internal signals can be found at byte 2+sizeof(long).

The first set of signals listed in the signals file are the externals in the following format:

```
-- External Signals...
```

```
--
```

```
[signal name] [direction/mode] [connector string]
```

The signal name is the corresponding signal name from the incoming VHDL structural file. The direction is one of the following single characters: i, o, b, where i=input, o=output and b=bidirectional. The mode is for power pins, which are designated with a "p". The second set of signals listed are the internals in the following format:

```
-- Internal signals
```

```
--
```

```
[signal name]
```

Once again, the signal name is the corresponding internal signal name from the incoming VHDL structural file. An example of the signals file for the example circuit and model (see APPENDIX C) is provided below.

```
--
```

```
--External Signals...
```

```
--
```

```
P1_1 i P1A.1
```

```
P1_2 i P1A.2
```

```
P1_3 i P1A.3
```

```
P1_4 i P1A.4
```

```
P1_5 i P1A.5
```

```
P1_6 o P1A.6
```

```
P1_7 i P1A.7
```

```
VCC p P1A.8
```

```
GND p P1A.9
```

```
--
```

```
-- Internal signals...
```

```
--
```

```
Y1_A3
```

```
Y2_P12
```

```
Y3_P4
```

```
Y4_P9
```

```
P7_B3
```

```
P10_B1_P5
```

```
P15_B2
```

The final output of the WAVES Testbench Writer tool is a WAVES compatible testbench that can be used to simulate the VHDL structural model and produce vectors that will eventually be used to test the actual board on an ATE. An example of the testbench created for the sample model in Figure C1.1 is shown in Figure C1.2.

Note the comments that are in **bold** letters in Figure C1.2. The WAVES compatible testbench created is based on certain rules and naming conventions that must be followed. The first concerns the naming of packages and functions that are contained in the WAVES dataset that are associated with the VHDL model. These rules are noted in bold lettering as comments in the initial use clauses of the test bench. Another rule that must be adhered to is making sure that the external signal declarations in the VHDL file are in the identical order as the columns contained in the WAVES external vector file. This is important as evidenced by the 'Translate' function defined in the test bench (this function is identified by bold letters in Figure C1.2). The translate function reads the logic values contained in the WAVES port list, translates the logic value to a std\_logic value, and applies it to a signal in the testbench. It is here where the signal names from the VHDL model are matched with the vectors contained in the WAVES external file. If the signals are not in the correct order, the simulation of the test bench may contain errors. Another rule concerns logic value names. The logic value names defined in the WAVES dataset **MUST** be the same names noted in Figure C1.2.

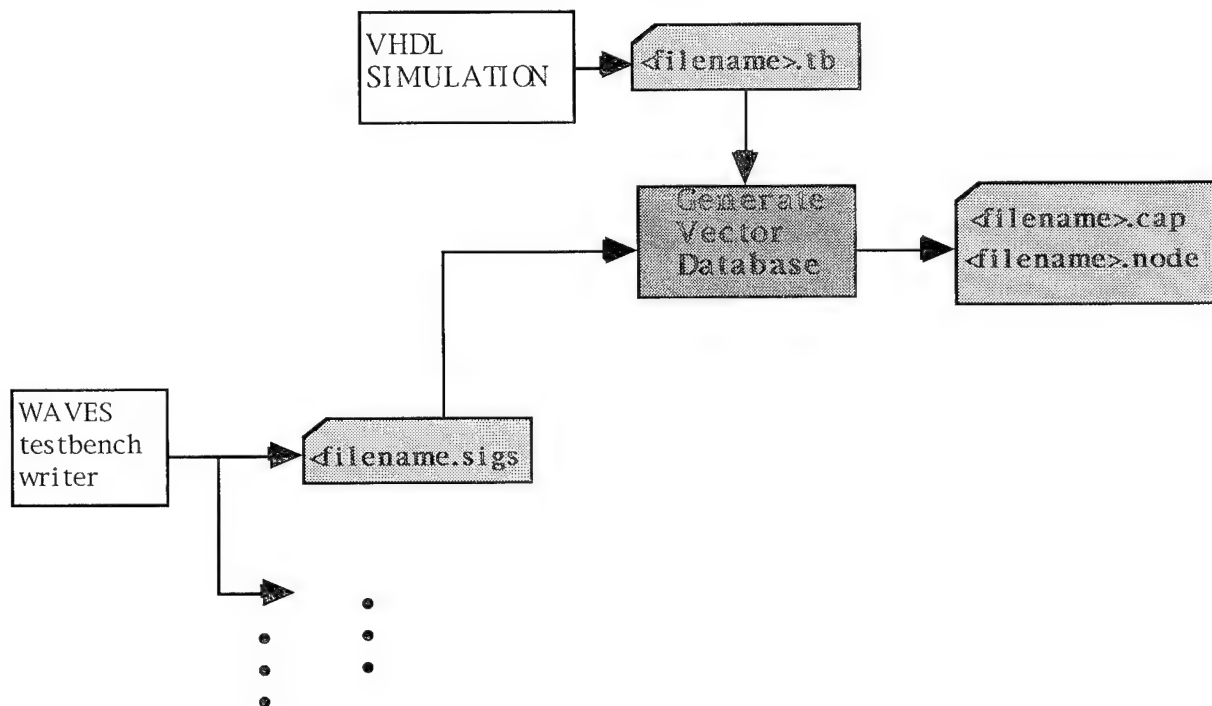
## 1.2 Vector Translation Tools

This tool is a GenRad specific translation tool designed to read the vector information produced via simulation of the VHDL testbench created by the WAVES Testbench Writer. Three files are created by this ID Workbench option in a two step process:

- step 1: execute "Generate Vector Database" tool
- step2: execute "GR2751 Translator (TGO)" tool

Step 1 above takes the following files as input: a file called <filename>.tb and a file called <filename>.sigs, where it is assumed that both the .tb and .sigs have the same filename (e.g., vidal.tb and vidal.sigs). The .tb file is the output file of the testbench simulation. The .sigs file is one of the outputs of the Generate WAVES Testbench tool described above. The output of the Generate Vector Database tool will be the GenRad .CAP file (see Figure B1) and another file called <filename>.nodes, where <filename> is the name of the file (without extension) given to the output of the testbench simulation and the .sigs file. Figure B2 below provides an explanation of this process.





**FIGURE B2: EXECUTION OF VECTOR TRANSLATION TOOL "Generate Vector Database"**

In the above figure, the <filename> is equivalent to the name of the VHDL structural file. **The name of the output file from simulation is currently <filename>.cap and will have to be renamed to <filename>.tb prior to running tb\_to\_nodes.** The .nodes file is the primary input to step 2 above, execute GR2751 Translator. This process creates the GenRad .TGO and GenRad .CTX files noted in Figure B1.

### 1.2.1 Execution

Refer to Section 3.2.2 for information on how to execute the ID Workbench Vector Translation Tools. Shown below is an example of the output message generated for the example circuit shown in Appendix C.

```

iitrisun% tb_to_nodes.x vidal
TISSS Tool TB_TO_NODES Version 2.2 as of 03-JUN-1994
Executing TB_TO_NODES on vidal
Nodes output file is vidal.nodes
Capture output file is vidal.cap
Reading .sigs file...
Number of VHDL signals = 18
Number of TGO I/O signals = 7
    (not counting new_slice, strobes, and power signals)
Reading .tb file to count slices...
Signals in .cap file will be encoded in base 94 using chars: !"#$%&'()*+,-
./0123
456789;<=>?@ABCDEFGHIJKLMNPOQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
mnopqrstuvwxyz{|}~
Number of slices = 28  NOTE: Contents of the last slice are ignored.
Reading .tb file to convert data...

```

There are two items of interest in the above output. First, the lines in italics show the output of the `tb_to_nodes` program which is executed as a result of choosing the Generate Vector Database option in the ID Workbench. The `.cap` file is the GenRad .CAP file shown in Figure B1. The `.nodes` file is the input for the `nodes_to_tgo.x` program, which is executed as part of the GR2751 Translator (TGO) option in the ID Workbench. The second item of interest is that in creating the `.nodes` file, the signal information associated with the `new_slice`, `strobe` and `power` signals does not get written to the `.nodes` file. This information is not required for creation of the GenRad .TGO file. The “`new_slice`” signal, as described in Reference [4], is used to inherit any cycle time information contained in the original simulation stimulus file. Also, the `strobe` signal is used to determine when in the .TGO file strobes are required. Power signal information is in the `<filename>.tb` file for creation of the GenRad .CAP file, which is used by the Genisys™ tools to create a probing database.

**APPENDIX C**  
**EXAMPLE FIGURES AND FILES**

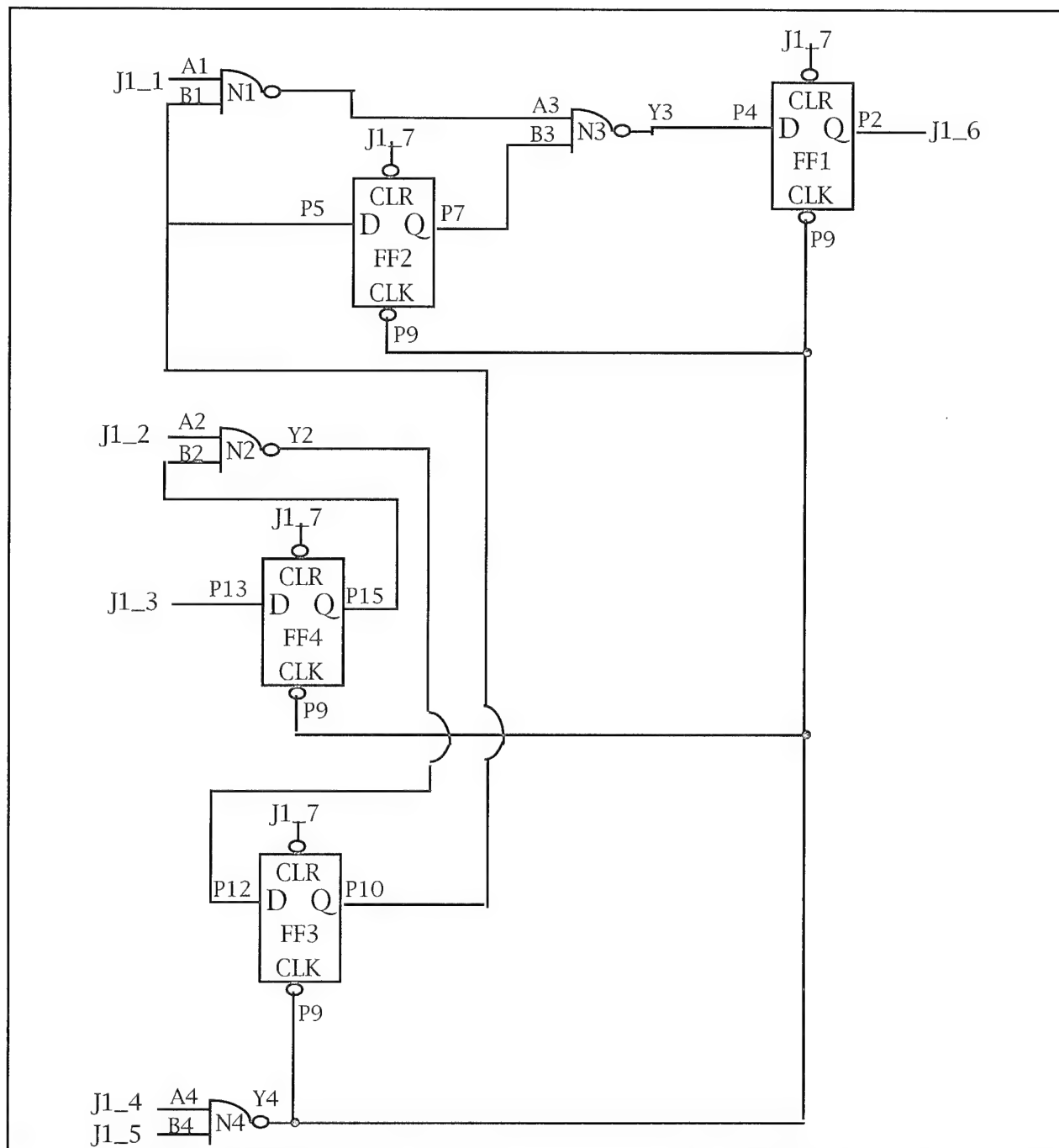


FIGURE C1. EXAMPLE CIRCUIT

```

--
-- Model For Vidal's Test Circuit
--
library ieee;
use ieee.std_logic_1164.all;

--
--
-- Declare Test Bench Entity.
-- T2-1 will extract only the external pins identified in the port
-- declaration below; any generics will be copied, as is, into the
-- test bench.
--
Entity VIDAL_MODEL is
port(
    P1_1 : in std_logic := 'X';
    P1_2 : in std_logic := 'X';
    P1_3 : in std_logic := 'X';
    P1_4 : in std_logic := 'X';
    P1_5 : in std_logic := 'X';
    P1_6 : out std_logic := 'X';
    P1_7 : in std_logic := 'X';
    VCC : in std_logic := '1';
    GND : in std_logic := '0');
--
-- Only signals of type std_logic or std_logic_vector are valid
-- signal types within the entity declaration in this version of
-- T2-1. T2-1 also requires that all logic vector signals be defined
-- with integer indices (variables and constants are not allowed).
-- Example:
-- a: in std_logic_vector ( 0 to 31 ); -- legal
-- b: in std_logic_vector ( x to y ); -- illegal
--

end VIDAL_MODEL;

```

FIGURE C1.1 VHDL STRUCTURAL DESCRIPTION FOR EXAMPLE CIRCUIT



```

--
-- Architecture of Test Bench
--
architecture STRUCTURE of VIDAL_MODEL is
--
-- T2-1 will extract all component blocks, signals and 'for'
-- statements that are within the above 'architecture' statement
-- and the 'begin' statement and put it into the appropriate place
-- in the test bench.
--

    signal Y1_A3 : std_logic := 'X';
    signal Y2_P12 : std_logic := 'X';
    signal Y3_P4 : std_logic := 'X';
    signal Y4_P9 : std_logic := 'X';

    signal P7_B3 : std_logic := 'X';
    signal P10_B1_P5 : std_logic := 'X';
    signal P15_B2 : std_logic := 'X';

component TTL00
port (
    A1 : in std_logic; -- pin 1
    B1 : in std_logic; -- pin 2
    Y1 : buffer std_logic; -- pin 3

    A2 : in std_logic; -- pin 4
    B2 : in std_logic; -- pin 5
    Y2 : buffer std_logic; -- pin 6

    A3 : in std_logic; -- pin 9
    B3 : in std_logic; -- pin 10
    Y3 : buffer std_logic; -- pin 8

```

**FIGURE C1.1: CONTINUED**

```

        A4 : in std_logic; -- pin 12
        B4 : in std_logic; -- pin 13
        Y4 : buffer std_logic; -- pin 11
        VCC : in std_logic := '1';
        GND : in std_logic := '0';
    );
end component;
--
-- Although shown here following the component
-- delcaration statement, the following 'for' statment can appear
-- anywhere within the file. For instance, a seperate
-- configuration section could be used containing all of the
-- needed 'for' statements.
--

    for all:TTL00 use entity work.TTL00(stru);

component TTL175
port(
    P1 : in std_logic; -- CLR
    P2 : buffer std_logic; -- Q1
    P3 : out std_logic; -- Q1B
    P4 : in std_logic; -- D1
    P5 : in std_logic; -- D2
    P6 : out std_logic; -- Q2B
    P7 : buffer std_logic; -- Q2
    GND: in std_logic := '0';
    P9 : in std_logic; -- CLK
    P10 : buffer std_logic; -- Q3
    P11 : out std_logic; -- Q3B
    P12 : in std_logic; -- D3
    P13 : in std_logic; -- D4
    P14 : out std_logic; -- Q4B
    P15 : buffer std_logic; -- Q4
    VCC: in std_logic := '1');

    end component;

    for all:TTL175 use entity work.TTL175(BEHAVIORAL);

```

**FIGURE C1.1: CONTINUED**

```

begin
--
-- The T2-1 tool will extract all generic maps and port maps
-- appearing between the 'begin' and 'end' statements.
--
U1 : TTL00
    PORT MAP(

        A1 => P1_1,
        B1 => P10_B1_P5,
        Y1 => Y1_A3,

        A2 => P1_2,
        B2 => P15_B2,
        Y2 => Y2_P12,

        A3 => Y1_A3,
        B3 => P7_B3,
        Y3 => Y3_P4,

        A4 => P1_4,
        B4 => P1_5,
        Y4 => Y4_P9,

        GND => GND,
        VCC => VCC);

U2 : TTL175
    PORT MAP(

        P1 => P1_7,
        P2 => P1_6,
        P3 => OPEN,

```

FIGURE C1.1: CONTINUED

-- any unused pins on a component **MUST** be designated as open as  
-- shown here. True 'no-connects' on a device do not have to be  
-- accounted for in the model. All other pins, including power  
-- pins, **MUST** be accounted for in the model (e.g., VCC and GND must  
-- be declared).

```
P4 => Y3_P4,  
P5 => P10_B1_P5,  
P6 => OPEN,  
P7 => P7_B3,  
GND => GND, -- pin 8  
P9 => Y4_P9,  
P10 => P10_B1_P5,  
P11 => OPEN,  
P12 => Y2_P12,  
P13 => P1_3,  
P14 => OPEN,  
P15 => p15_B2,  
VCC => VCC); -- pin 16
```

END STRUCTURE;

FIGURE C1.1: CONTINUED

```

--
-- Filename : tb_vidal.VHD
-- Created  : 16-Jun-1994 13:51:42
--
-- Created by: T2-1 Version 1.1
--
--      IIT ResearchInstitute
--      201 Mill Street
--      Rome, NY 13440
--      (315) 339-7119
--      runkle@mail.iitri.com
--      wswavely@mail.iitri.com
--      jbeaton@mail.iitri.com
--
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.all;
USE WORK.TC_WAVES_LOGIC.all;      -- TC will be the prefix of TISSS dataset
USE WORK.WAVES_OBJECTS.all;
USE WORK.WAVEFORM_GENERATOR.all;  -- The package in the dataset MUST
USE STD.TEXTIO.all;              -- BE called WAVEFORM_GENERATOR.

--
-- Declare Test Bench Entity
--
Entity TEST_BENCH is
end;

--
-- Architecture of Test Bench
--
architecture WAVES_APP of TEST_BENCH is

--
-- The external pins from the port map
--
SIGNAL P1_1 : STD_LOGIC := 'X';
SIGNAL P1_2 : STD_LOGIC := 'X';
SIGNAL P1_3 : STD_LOGIC := 'X';
SIGNAL P1_4 : STD_LOGIC := 'X';
SIGNAL P1_5 : STD_LOGIC := 'X';
SIGNAL W_P1_6 : STD_LOGIC := 'X';
SIGNAL P1_6 : STD_LOGIC := 'X';
SIGNAL P1_7 : STD_LOGIC := 'X';
SIGNAL VCC : STD_LOGIC := '1';
SIGNAL GND : STD_LOGIC := '0';

```

**FIGURE C1.2: SAMPLE WAVES COMPATIBLE VHDL TESTBENCH PRODUCED BY THE “WAVES Testbench Writer” TOOL**



```

--
-- The internals pins
--
SIGNAL Y1_A3 : STD_LOGIC := 'X';
SIGNAL Y2_P12 : STD_LOGIC := 'X';
SIGNAL Y3_P4 : STD_LOGIC := 'X';
SIGNAL Y4_P9 : STD_LOGIC := 'X';
SIGNAL P7_B3 : STD_LOGIC := 'X';
SIGNAL P10_B1_P5 : STD_LOGIC := 'X';
SIGNAL P15_B2 : STD_LOGIC := 'X';
SIGNAL NEW_SLICE : STD_LOGIC;

SIGNAL WAVES_DATA : WAVES_PORT_LIST;

--
-- The components were lifted from the model
--
component TTL00
port (
    A1 : in std_logic;
    B1 : in std_logic;
    Y1 : buffer std_logic;

    A2 : in std_logic;
    B2 : in std_logic;
    Y2 : buffer std_logic;

    A3 : in std_logic;
    B3 : in std_logic;
    Y3 : buffer std_logic;

    A4 : in std_logic;
    B4 : in std_logic;
    Y4 : buffer std_logic;
    VCC: in std_logic := '1';
    GND: in std_logic := '0' );
end component;

```

FIGURE C1.2. CONTINUED

```

component TTL175
port(
    P1 : in std_logic;
    P2 : buffer std_logic;
    P3 : out std_logic;
    P4 : in std_logic;
    P5 : in std_logic;
    P6 : out std_logic;
    P7 : buffer std_logic;
    GND : in std_logic := '0';
    P9 : in std_logic;
    P10 : buffer std_logic;
    P11 : out std_logic;
    P12 : in std_logic;
    P13 : in std_logic;
    P14 : out std_logic;
    P15 : buffer std_logic;
    VCC : in std_logic := '1');
end component;

for all:TTL00 use entity work.TTL00(stru);
for all:TTL175 use entity work.TTL175(BEHAVIORAL);

--
-- Define a file type for the CAPTURE data
--
file CAPTURE_FILE : TEXT is out "tc_cap.out";

--
-- Convert STD_LOGIC to a character
--
function TO_CHAR( VALUE : std_logic; DIRECT : character; DRIVE_VALUE : logic_value )
return character is
begin
    if VALUE = 'X' then
        case DRIVE_VALUE is
            when DRIVE_0 => return 'C';
            when DRIVE_1 => return 'D';
            when others => return 'X';
        end case;
    else
        if DIRECT = 'I' then
            case VALUE is
                when '0' => return '0';
                when '1' => return '1';
                when 'Z' => return 'U';
                when 'U' => return 'U';
                when '-' => return '-';
            end case;
        end if;
    end if;
end function;

```

FIGURE C1.2. CONTINUED

```

else -- DIRECT = 'O'
case VALUE is
  when 'L' => return 'L';
  when 'H' => return 'H';
  when '0' => return 'L';
  when '1' => return 'H';
  when 'Z' => return 'U';
  when 'U' => return 'U';
  when '-' => return '-';
end case;
end if;
end if;

end TO_CHAR;

function TO_CHAR( VALUE : in std_logic; DIRECT : character ) return character is
begin

if DIRECT = 'I' then
case VALUE is
  when '0' => return '0';
  when '1' => return '1';
  when 'X' => return 'X';
  when 'Z' => return 'Z';
  when 'U' => return 'U';
  when '-' => return '-';
end case;
else -- DIRECT = 'O'
case VALUE is
  when 'L' => return 'L';
  when 'H' => return 'H';
  when '0' => return 'L';
  when '1' => return 'H';
  when 'X' => return 'X';
  when 'Z' => return 'X';
  when 'U' => return 'U';
  when '-' => return '-';
end case;
end if;

end TO_CHAR;

--
-- These LOGIC_VALUES will be the standard names and must be the
-- same in all TISSS WAVES DATASETS: UNKNOWN, UN_INT, HI_IMP,
-- DONT_CARE, DRIVE_0, DRIVE_1, SENSE_0, and SENSE_1.
--

function To_1164( VALUE : in Logic_value ) return Std_logic is
begin

```

FIGURE C1.2. CONTINUED

```

case VALUE is
  when UNKNOWN => return '-';
  when UN_INT  => return 'U';
  when HI_IMP  => return 'Z';
  when DONT_CARE => return 'X';
  when DRIVE_0 => return '0';
  when DRIVE_1 => return '1';
  when SENSE_0 => return '0';
  when SENSE_1 => return '1';
end case;

end To_1164;
--
-- Determine if WAVES is driving model, if not drive Z.
--
function TO_DRIVE( VALUE2 : Logic_value ) return Std_logic is
begin
  case VALUE2 is
    when UNKNOWN => return 'Z';
    when UN_INT  => return 'Z';
    when HI_IMP  => return 'Z';
    when DONT_CARE => return 'Z';
    when DRIVE_0 => return '0';
    when DRIVE_1 => return '1';
    when SENSE_0 => return 'Z';
    when SENSE_1 => return 'Z';
  end case;

  end TO_DRIVE;
--
-- Determine direction of signal for captured output (BIDIRECTIONAL I/O)
--

function FIND_DIRECT( value3 : Logic_value ) return character is
begin
  case value3 is
    when UNKNOWN => return 'O';
    when UN_INT  => return 'O';
    when HI_IMP  => return 'O';
    when DONT_CARE => return 'O';
    when DRIVE_0  => return 'I';
    when DRIVE_1  => return 'I';
    when SENSE_0  => return 'O';
    when SENSE_1  => return 'O';
  end case;

  end FIND_DIRECT;

```

FIGURE C1.2. CONTINUED

```

--
-- Define the translate function
--
begin
  WAVES : Waveform ( WAVES_DATA );

TRANSLATE:
  process( WAVES_DATA )
  begin

    P1_1 <= To_1164(Logic_value'val(WAVES_DATA.WPL(1).L_VALUE));
    P1_2 <= To_1164(Logic_value'val(WAVES_DATA.WPL(2).L_VALUE));
    P1_3 <= To_1164(Logic_value'val(WAVES_DATA.WPL(3).L_VALUE));
    P1_4 <= To_1164(Logic_value'val(WAVES_DATA.WPL(4).L_VALUE));
    P1_5 <= To_1164(Logic_value'val(WAVES_DATA.WPL(5).L_VALUE));
    W_P1_6 <= To_1164(Logic_value'val(WAVES_DATA.WPL(6).L_VALUE));
    P1_7 <= To_1164(Logic_value'val(WAVES_DATA.WPL(7).L_VALUE));
    VCC <= To_1164(Logic_value'val(WAVES_DATA.WPL(8).L_VALUE));
    GND <= To_1164(Logic_value'val(WAVES_DATA.WPL(9).L_VALUE));
    NEW_SLICE <= To_1164(Logic_value'val(WAVES_DATA.WPL(10).L_VALUE));
  end process;

U1 : TTL00
PORT MAP( A1 => P1_1,
          B1 => P10_B1_P5,
          Y1 => Y1_A3,
          A2 => P1_2,
          B2 => P15_B2,
          Y2 => Y2_P12,
          A3 => Y1_A3,
          B3 => P7_B3,
          Y3 => Y3_P4,
          A4 => P1_4,
          B4 => P1_5,
          Y4 => Y4_P9,
          VCC => VCC,
          GND => GND);

U2 : TTL175
PORT MAP( P1 => P1_7,
          P2 => P1_6,
          P3 => open,
          P4 => Y3_P4,
          P5 => P10_B1_P5,
          P6 => open,
          P7 => P7_B3,
          GND => GND,

```

FIGURE C1.2. CONTINUED



```

P9 => Y4_P9,
P10 => P10_B1_P5,
P11 => open,
P12 => Y2_P12,
P13 => P1_3,
P14 => open,
P15 => p15_B2,
VCC => VCC);

--
-- Write CAPTURE data to a disk file
--
CAP_P1_1 :
  process( P1_1 )
    variable DIRECT : character := 'T';
    variable LOUT : line;
    variable SIG_NAME : string (1 to 4) := "P1_1";
    variable STATE : character;
    variable SIM_TIME : time;
    variable SPACE : character := ' ';
  begin
    STATE := TO_CHAR(P1_1,DIRECT);
    SIM_TIME := now;
    write(LOUT, SIG_NAME);
    write(LOUT, SPACE);
    write(LOUT, STATE);
    write(LOUT, SPACE);
    write(LOUT, SIM_TIME/ns);
    writeline(CAPTURE_FILE, LOUT);
  end process;

CAP_P1_2 :
  process( P1_2 )
    variable DIRECT : character := 'T';
    variable LOUT : line;
    variable SIG_NAME : string (1 to 4) := "P1_2";
    variable STATE : character;
    variable SIM_TIME : time;
    variable SPACE : character := ' ';
  begin
    STATE := TO_CHAR(P1_2,DIRECT);
    SIM_TIME := now;
    write(LOUT, SIG_NAME);
    write(LOUT, SPACE);
    write(LOUT, STATE);
    write(LOUT, SPACE);
    write(LOUT, SIM_TIME/ns);
    writeline(CAPTURE_FILE, LOUT);
  end process;

```

FIGURE C1.2. CONTINUED

```

CAP_P1_3:
  process( P1_3 )
    variable DIRECT : character := 'T';
    variable LOUT : line;
    variable SIG_NAME : string (1 to 4) := "P1_3";
    variable STATE : character;
    variable SIM_TIME : time;
    variable SPACE : character := ' ';
  begin
    STATE := TO_CHAR(P1_3,DIRECT);
    SIM_TIME := now;
    write(LOUT, SIG_NAME);
    write(LOUT, SPACE);
    write(LOUT, STATE);
    write(LOUT, SPACE);
    write(LOUT, SIM_TIME/ns);
    writeline(CAPTURE_FILE, LOUT);
  end process;

```

```

CAP_P1_4:
  process( P1_4 )
    variable DIRECT : character := 'T';
    variable LOUT : line;
    variable SIG_NAME : string (1 to 4) := "P1_4";
    variable STATE : character;
    variable SIM_TIME : time;
    variable SPACE : character := ' ';
  begin
    STATE := TO_CHAR(P1_4,DIRECT);
    SIM_TIME := now;
    write(LOUT, SIG_NAME);
    write(LOUT, SPACE);
    write(LOUT, STATE);
    write(LOUT, SPACE);
    write(LOUT, SIM_TIME/ns);
    writeline(CAPTURE_FILE, LOUT);
  end process;

```

```

CAP_P1_5:
  process( P1_5 )
    variable DIRECT : character := 'T';
    variable LOUT : line;
    variable SIG_NAME : string (1 to 4) := "P1_5";
    variable STATE : character;
    variable SIM_TIME : time;
    variable SPACE : character := ' ';

```

FIGURE C1.2. CONTINUED

```

begin
    STATE := TO_CHAR(P1_5,DIRECT);
    SIM_TIME := now;
    write(LOUT, SIG_NAME);
    write(LOUT, SPACE);
    write(LOUT, STATE);
    write(LOUT, SPACE);
    write(LOUT, SIM_TIME/ns);
    writeline(CAPTURE_FILE, LOUT);
end process;

CAP_P1_6 :
    process( P1_6 )
        variable VALUE3: Logic_value
        variable DIRECT : character := 'O';
        variable LOUT : line;
        variable SIG_NAME : string (1 to 4) := "P1_6";
        variable STATE : character;
        variable SIM_TIME : time;
        variable SPACE : character := ' ';
    begin
        VALUE3 := Logic_value'val(WAVES_DATA.WPL(6).L_VALUE);
        DIRECT := FIND_DIRECT (VALUE3);
        STATE := TO_CHAR(P1_6,DIRECT, VALUE 3);
        SIM_TIME := now;
        write(LOUT, SIG_NAME);
        write(LOUT, SPACE);
        write(LOUT, STATE);
        write(LOUT, SPACE);
        write(LOUT, SIM_TIME/ns);
        writeline(CAPTURE_FILE, LOUT);
    end process;

CAP_P1_6_STROBE :
    process( W_P1_6 )
        variable VALUE3 : Logic_value;
        variable DIRECT : character;
        variable LOUT : line;
        variable SIG_NAME : string (1 to 5) := "*P1_6";
        variable STATE : character;
        variable SIM_TIME : time;
        variable SPACE : character := ' ';
    begin
        STATE := TO_CHAR(W_P1_6,DIRECT);
        VALUE3 := Logic_value'val(WAVES_DATA.WPL(6).L_VALUE);
        DIRECT := FIND_DIRECT (VALUE3);
        IF STATE = 'H' or STATE = 'L' or STATE = 'X' then
            STATE := '1';
        ELSE
            STATE := '0';
        end if;
    end process;

```

FIGURE C1.2. CONTINUED

```

SIM_TIME := now;
write(LOUT, SIG_NAME);
write(LOUT, SPACE);
write(LOUT, STATE);
write(LOUT, SPACE);
write(LOUT, SIM_TIME/ns);
writeline(CAPTURE_FILE, LOUT);
if P1_6'stable then
  SIG_NAME := "P1_6 ";
  STATE := TO_CHAR(P1_6, DIRECT);
  SIM_TIME := now;
  write(LOUT, SIG_NAME);
  write(LOUT, STATE);
  write(LOUT, SPACE);
  write(LOUT, SIM_TIME/ns);
  writeline(CAPTURE_FILE, LOUT);
end if;
end process;

CAP_P1_7 :
  process( P1_7 )
    variable DIRECT : character := 'T';
    variable LOUT : line;
    variable SIG_NAME : string (1 to 4) := "P1_7";
    variable STATE : character;
    variable SIM_TIME : time;
    variable SPACE : character := ' ';
  begin
    STATE := TO_CHAR(P1_7,DIRECT);
    SIM_TIME := now;
    write(LOUT, SIG_NAME);
    write(LOUT, SPACE);
    write(LOUT, STATE);
    write(LOUT, SPACE);
    write(LOUT, SIM_TIME/ns);
    writeline(CAPTURE_FILE, LOUT);
  end process;

CAP_VCC :
  process( VCC )
    variable DIRECT : character := 'T';
    variable LOUT : line;
    variable SIG_NAME : string (1 to 3) := "VCC";
    variable STATE : character;
    variable SIM_TIME : time;
    variable SPACE : character := ' ';

```

FIGURE C1.2. CONTINUED

```

begin
  STATE := TO_CHAR(VCC,DIRECT);
  SIM_TIME := now;
  write(LOUT, SIG_NAME);
  write(LOUT, SPACE);
  write(LOUT, STATE);
  write(LOUT, SPACE);
  write(LOUT, SIM_TIME/ns);
  writeline(CAPTURE_FILE, LOUT);
end process;

```

CAP\_GND :

```

process( GND )
  variable DIRECT : character := 'T';
  variable LOUT : line;
  variable SIG_NAME : string (1 to 3) := "GND";
  variable STATE : character;
  variable SIM_TIME : time;
  variable SPACE : character := ' ';
begin
  STATE := TO_CHAR(GND,DIRECT);
  SIM_TIME := now;
  write(LOUT, SIG_NAME);
  write(LOUT, SPACE);
  write(LOUT, STATE);
  write(LOUT, SPACE);
  write(LOUT, SIM_TIME/ns);
  writeline(CAPTURE_FILE, LOUT);
end process;

```

CAP\_Y1\_A3 :

```

process( Y1_A3 )
  variable DIRECT : character := 'O';
  variable LOUT : line;
  variable SIG_NAME : string (1 to 5) := "Y1_A3";
  variable STATE : character;
  variable SIM_TIME : time;
  variable SPACE : character := ' ';
begin
  STATE := TO_CHAR(Y1_A3,DIRECT);
  SIM_TIME := now;
  write(LOUT, SIG_NAME);
  write(LOUT, SPACE);
  write(LOUT, STATE);
  write(LOUT, SPACE);
  write(LOUT, SIM_TIME/ns);
  writeline(CAPTURE_FILE, LOUT);
end process;

```

FIGURE C1.2. CONTINUED

```

CAP_Y2_P12 :
  process( Y2_P12 )
    variable DIRECT : character := 'O';
    variable LOUT : line;
    variable SIG_NAME : string (1 to 6) := "Y2_P12";
    variable STATE : character;
    variable SIM_TIME : time;
    variable SPACE : character := ' ';
  begin
    STATE := TO_CHAR(Y2_P12,DIRECT);
    SIM_TIME := now;
    write(LOUT, SIG_NAME);
    write(LOUT, SPACE);
    write(LOUT, STATE);
    write(LOUT, SPACE);
    write(LOUT, SIM_TIME/ns);
    writeline(CAPTURE_FILE, LOUT);
  end process;

```

```

CAP_Y3_P4 :
  process( Y3_P4 )
    variable DIRECT : character := 'O';
    variable LOUT : line;
    variable SIG_NAME : string (1 to 5) := "Y3_P4";
    variable STATE : character;
    variable SIM_TIME : time;
    variable SPACE : character := ' ';
  begin
    STATE := TO_CHAR(Y3_P4,DIRECT);
    SIM_TIME := now;
    write(LOUT, SIG_NAME);
    write(LOUT, SPACE);
    write(LOUT, STATE);
    write(LOUT, SPACE);
    write(LOUT, SIM_TIME/ns);
    writeline(CAPTURE_FILE, LOUT);
  end process;

```

```

CAP_Y4_P9 :
  process( Y4_P9 )
    variable DIRECT : character := 'O';
    variable LOUT : line;
    variable SIG_NAME : string (1 to 5) := "Y4_P9";
    variable STATE : character;
    variable SIM_TIME : time;
    variable SPACE : character := ' ';

```

FIGURE C1.2. CONTINUED



```

begin
  STATE := TO_CHAR(Y4_P9,DIRECT);
  SIM_TIME := now;
  write(LOUT, SIG_NAME);
  write(LOUT, SPACE);
  write(LOUT, STATE);
  write(LOUT, SPACE);
  write(LOUT, SIM_TIME/ns);
  writeline(CAPTURE_FILE, LOUT);
end process;

CAP_P7_B3 :
  process( P7_B3 )
    variable DIRECT : character := 'O';
    variable LOUT : line;
    variable SIG_NAME : string (1 to 5) := "P7_B3";
    variable STATE : character;
    variable SIM_TIME : time;
    variable SPACE : character := ' ';
  begin
    STATE := TO_CHAR(P7_B3,DIRECT);
    SIM_TIME := now;
    write(LOUT, SIG_NAME);
    write(LOUT, SPACE);
    write(LOUT, STATE);
    write(LOUT, SPACE);
    write(LOUT, SIM_TIME/ns);
    writeline(CAPTURE_FILE, LOUT);
  end process;

CAP_P10_B1_P5 :
  process( P10_B1_P5 )
    variable DIRECT : character := 'O';
    variable LOUT : line;
    variable SIG_NAME : string (1 to 9) := "P10_B1_P5";
    variable STATE : character;
    variable SIM_TIME : time;
    variable SPACE : character := ' ';
  begin
    STATE := TO_CHAR(P10_B1_P5,DIRECT);
    SIM_TIME := now;
    write(LOUT, SIG_NAME);
    write(LOUT, SPACE);
    write(LOUT, STATE);
    write(LOUT, SPACE);-
    write(LOUT, SIM_TIME/ns);
    writeline(CAPTURE_FILE, LOUT);
  end process;

```

FIGURE C1.2. CONTINUED

```

CAP_P15_B2 :
  process( P15_B2 )
    variable DIRECT : character := 'O';
    variable LOUT : line;
    variable SIG_NAME : string (1 to 6) := "P15_B2";
    variable STATE : character;
    variable SIM_TIME : time;
    variable SPACE : character := ' ';
  begin
    STATE := TO_CHAR(P15_B2,DIRECT);
    SIM_TIME := now;
    write(LOUT, SIG_NAME);
    write(LOUT, SPACE);
    write(LOUT, STATE);
    write(LOUT, SPACE);
    write(LOUT, SIM_TIME/ns);
    writeline(CAPTURE_FILE, LOUT);
  end process;

CAP_NEW_SLICE :
  process( NEW_SLICE )
    variable DIRECT : character := 'O';
    variable LOUT : line;
    variable SIG_NAME : string (1 to 9) := "NEW_SLICE";
    variable STATE : character;
    variable SIM_TIME : time;
    variable SPACE : character := ' ';
  begin
    STATE := TO_CHAR(NEW_SLICE,DIRECT);
    SIM_TIME := now;
    write(LOUT, SIG_NAME);
    write(LOUT, SPACE);
    write(LOUT, STATE);
    write(LOUT, SPACE);
    write(LOUT, SIM_TIME/ns);
    writeline(CAPTURE_FILE, LOUT);
  end process;

END waves_app;

```

FIGURE C1.2. CONTINUED

***MISSION  
OF  
ROME LABORATORY***

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

- a. Conducts vigorous research, development and test programs in all applicable technologies;
- b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;
- c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;
- d. Promotes transfer of technology to the private sector;
- e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.